



**Government Acquisition  
Through  
Electronic Commerce**

**(GATEC)**

**Internal Description and  
Maintenance Guide**

**Doc Id:** TISP940106

**Rev Id:** Release 1

**Release Date:** 1 January 1994

Prepared for:

Aeronautical Systems Center  
Operational and Central Support  
Contracting Division  
Air Force Materiel Command  
Wright-Patterson AFB, Ohio

Prepared by:

LLNL GATEC Project Staff  
EC/EDI Projects  
Technology Information Systems Program  
Lawrence Livermore National Laboratory  
Livermore, CA 94550



This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

Work performed under the auspices of the United States Department of Energy by Lawrence Livermore National Laboratory under contract number W-7405-ENG-48 and DOE Work For Others project number L-788A.

---



---

## Table of Contents

---

SECTION 1	Interface to Legacy System.....	1
1.1	BCAS Download of RFQs.....	1
	1.1.1.1 getopr_bsp_cron3 .....	1
1.1.2	Support Software for RFQ download .....	5
	1.1.2.1 readopr2 .....	5
1.2	Download of other BCAS specific Data .....	6
	1.2.1.1 getstmntship_cron.....	6
1.3	Utilities Used for Download of BCAS Data .....	8
	1.3.1.1 getwangfiles .....	8
	1.3.1.2 BCASrunproc .....	9
1.4	BCAS Award Upload.....	12
	1.4.1.1 putuploads_cron.....	12
	1.4.1.2 BCASupload .....	15
1.4.2	Utilities Used in Upload of BCAS Data .....	20
	1.4.2.1 setUTN_aw_to_cl .....	20
	1.4.2.2 get_UTNNumber_from_cdf .....	20
	1.4.2.3 get_piin_from_cdf .....	21
	1.4.2.4 acqerr .....	21
1.4.3	Gateway Utilities Used in Upload of Award Data...22	
	1.4.3.1 resp_err, back2AWD01, backout, checkvars, dumpvars, menumove, parsefields, officetosysadm, upload, walkmenus .....	22
1.5	BCAS Item Description Upload .....	24
	1.5.1.1 upload_bcas_item_desc_cron .....	24
	1.5.1.2 BCASitemupload.....	26
	1.5.1.3 selitemcdf.....	37
	1.5.1.4 itemerrtouser .....	37
1.6	BCAS Cancel Award Upload .....	38
	1.6.1.1 upload_bcas_cancel_award_cron .....	38
	1.6.1.2 BCAScancelaward.....	41
	1.6.1.3 selcancelcdf.....	52
	1.6.1.4 cancelerrtouser .....	52
1.7	Support Software for Upload/Download Crons.....	53
	1.7.1.1 errtomgr .....	53
	1.7.1.2 errtouser .....	53
	1.7.1.3 errtomgr .....	54
	1.7.1.4 cdf_check .....	54
	1.7.1.5 cdf_regexp .....	55
	1.7.1.6 regcomp, regexec, regsub, regerror .....	56
1.8	Gateway Support Software .....	59

---

---

## Table of Contents (Continued)

---

	1.8.1.1	bcaslogin, bcaslogout, checkresp, finis, pfkey, substr, fixfield, cmdargs, expresslogout .....	59
1.9	Multi-User Queue Public Domain Software .....		61
	1.9.1.1	q .....	61
	1.9.1.2	db_open, db_close, db_store, db_fetch, db_delete, db_rewind, db_nextrec .....	63
	1.9.1.3	tisp.....	64
1.10	Compilation and Installation of Interface to Legacy System Software .....		65
1.11	Miscellaneous Software .....		68
	1.11.1	Acknowledgment Monitoring Software.....	68
	1.11.1.1	ack_cron_pl.....	68
	1.11.1.2	997CDFtoDB .....	69
SECTION 2	Distributed User Interface (DUI) Software .....		73
2.1	DUI Toolkit.....		73
	2.1.1	Basic Architecture .....	74
	2.1.1.1	Server .....	74
	2.1.1.2	Client.....	75
	2.1.2	Communications .....	75
	2.1.2.1	Communications link.....	75
	2.1.2.2	DUI Protocol.....	76
	2.1.2.3	A Typical Session .....	76
	2.1.3	DUI Class Hierarchy and Libraries .....	77
	2.1.4	Application Programming Guide .....	79
	2.1.4.1	Beginning and Ending a session .....	79
	2.1.4.2	Event Driven Programming .....	79
	2.1.4.3	Creating Forms .....	80
	2.1.4.4	Modifiers and Constraints.....	81
	2.1.4.5	Callback Functions .....	81
	2.1.5	Code Generation .....	83
	2.1.5.1	Client Code Generation .....	83
	2.1.6	DUI Source Directory .....	84
	2.1.7	Detailed Working Example.....	85
	2.1.8	DUI Detail Class/Object Descriptions .....	90
	2.1.8.1	AppControl.....	91
	2.1.8.2	ChannelBuf .....	93
	2.1.8.3	Communication_Object .....	94
	2.1.8.4	Constraint .....	96
	2.1.8.5	ConfigInfo .....	97

---

---

## Table of Contents

---

	2.1.8.6	DUI.....	100
	2.1.8.7	DUI_Command .....	102
	2.1.8.8	DUI_Component .....	105
	2.1.8.9	DUI_Dialog.....	107
	2.1.8.10	DUI_End_Command .....	109
	2.1.8.11	DUI_Field .....	111
	2.1.8.12	DUI_Form .....	114
	2.1.8.13	DUI_Group .....	115
	2.1.8.14	DUI_Invisible_Field .....	117
	2.1.8.15	DUI_Label .....	119
	2.1.8.16	DUI_Multi_Selection.....	120
	2.1.8.17	DUI_Range .....	123
	2.1.8.18	DUI_Selection.....	125
	2.1.8.19	DUI_Table .....	128
	2.1.8.20	DUI_Text .....	132
	2.1.8.21	DUI_Toggle .....	134
	2.1.8.22	DUI_View .....	136
	2.1.8.23	DUI_Widget.....	139
	2.1.8.24	Date .....	141
	2.1.8.25	Filebuf_With_Audit .....	142
	2.1.8.26	Integer .....	144
	2.1.8.27	Justified .....	145
	2.1.8.28	Left_Justified .....	146
	2.1.8.29	Lower_Case .....	147
	2.1.8.30	Mandatory .....	148
	2.1.8.31	Military_Date .....	149
	2.1.8.32	Modifier .....	150
	2.1.8.33	Numeric.....	151
	2.1.8.34	Precision.....	152
	2.1.8.35	Regular_Expression .....	153
	2.1.8.36	Right_Justified .....	155
	2.1.8.37	STRING .....	156
	2.1.8.38	Session .....	160
	2.1.8.40	Table_Column.....	166
	2.1.8.41	Truncated .....	169
	2.1.8.42	Unjustified.....	170
	2.1.8.43	Upper_Case .....	171
2.2		GATEC Application .....	172
	2.2.1	Class Hierarchy .....	172
	2.2.2	Programming Hints .....	173
	2.2.3	GATEC DUI Source Tree.....	173
	2.2.4	GATEC Form Classes.....	173
		2.2.4.1 Award_Form .....	175

---

---

## Table of Contents (Continued)

---

	2.2.4.2	Compose_Message_Form .....	180
	2.2.4.3	Flag_Selection .....	183
	2.2.4.4	Message_Form .....	186
	2.2.4.5	Quote_Abstract_Form .....	191
	2.2.4.6	RFQ_Category .....	196
	2.2.4.7	Review_Quote_Form .....	199
	2.2.4.8	Review_RFQ_Form .....	203
	2.2.4.9	Vendor_Performance_Data .....	210
	2.2.4.10	Workload_Form .....	211
2.3		Lead Buyer Application .....	214
	2.3.1	Class Hierarchy .....	214
	2.3.2	Programming Hints .....	215
	2.3.3	Lead Buyer Source Tree .....	215
	2.3.4	Lead Buyer Form Classes .....	216
	2.3.4.1	Change_RFQs_Form .....	217
	2.3.4.2	List_RFQs_Form .....	219
	2.3.4.3	Price_History_Form .....	222
	2.3.4.4	Price_Performance_Form .....	224
	2.3.4.5	RFQ_Summary .....	226
	2.3.4.6	Range_List .....	230
	2.3.4.7	Select_RFQs_Form .....	232
	2.3.4.8	Statistics_Form .....	234
	2.3.4.9	Sort_Order .....	236
	2.3.4.10	String .....	238
	2.3.4.11	Summarized_RFQ .....	241
2.4		System Parameters Application .....	247
	2.4.1	Class Hierarchy .....	247
	2.4.2	Programming Hints .....	247
	2.4.3	System Parameters Source Tree .....	248
	2.4.4	System Parameters From Classes .....	248
	2.4.4.1	System_Parameters_Form .....	249
2.5		Windui Application .....	251
	2.5.1	Basic Implementation Strategy .....	252
	2.5.2	Class Hierarchy .....	252
	2.5.3	DUI Resources .....	254
	2.5.4	Communications .....	257
	2.5.5	WINDUI Source Directory .....	258
	2.5.6	WINDUI Classes .....	259
	2.5.6.1	Communications_Script .....	260
	2.5.6.2	Device_Independent_Bitmap .....	263
	2.5.6.3	Local_Atom .....	265
	2.5.6.4	Prompt_Dialog .....	267
	2.5.6.5	Pushbutton_Bitmap .....	269

---

---

## Table of Contents

---

	2.5.6.6 TMainWindow .....	271
	2.5.6.7 Table_String .....	273
	2.5.6.8 WTButton .....	275
	2.5.6.9 WTCheckBox .....	277
	2.5.6.10 WTComboBox .....	279
	2.5.6.11 WTEdit .....	281
	2.5.6.12 WTGroupBox .....	283
	2.5.6.13 WTListBox .....	284
	2.5.6.14 WTRadioButton .....	286
	2.5.6.15 WTStatic .....	288
	2.5.6.16 WTText .....	290
	2.5.6.17 WTWindow .....	292
	2.5.6.18 WTable .....	294
	2.5.6.19 w_Command .....	300
	2.5.6.20 w_Component .....	303
	2.5.6.21 w_End_Component .....	304
	2.5.6.22 w_Field .....	305
	2.5.6.23 w_Group .....	307
	2.5.6.24 w_Label .....	310
	2.5.6.25 w_Selection .....	312
	2.5.6.26 w_Table .....	314
	2.5.6.27 w_Text .....	317
	2.5.6.28 w_Toggle .....	319
	2.5.6.29 w_View .....	321
	2.5.6.30 w_Widget .....	323
	2.5.6.31 Session .....	329
	2.5.6.32 SerialBuf .....	332
SECTION 3	The GATEC Database Software .....	335
	3.0.1 NARQ & NORA .....	335
3.1	NORA Principles .....	336
	3.1.1 NORA Classes .....	336
	3.1.2 Limitations .....	342
	3.1.3 Detailed NORA Class Descriptions .....	342
	3.1.3.1 CharColumn .....	343
	3.1.3.2 Column .....	345
	3.1.3.3 DBObject .....	347
	3.1.3.4 Connection .....	350
	3.1.3.5 Database .....	352
	3.1.3.6 Condition .....	354

---

---

## Table of Contents (Continued)

---

	3.1.3.7 Dual .....	357
	3.1.3.8 ComplexQuery .....	358
	3.1.3.9 DateColumn .....	361
	3.1.3.10 Expression .....	364
	3.1.3.11 FetchedGroup .....	367
	3.1.3.12 FetchedRows .....	368
	3.1.3.13 FloatColumn .....	369
	3.1.3.14 ImmediateQuery .....	371
	3.1.3.15 Join .....	373
	3.1.3.16 LongColumn .....	375
	3.1.3.17 QueryResult .....	377
	3.1.3.18 RowID .....	379
	3.1.3.19 Sequence .....	381
	3.1.3.20 SimpleQuery .....	382
	3.1.3.21 Table .....	384
	3.1.3.22 RawColumn .....	387
	3.1.3.23 Query .....	389
	3.1.3.24 NumberColumn .....	391
3.2	NARQ Library Principles .....	393
3.2.1	NARQ Library Generation .....	393
	3.2.1.1 Acquisition Object .....	396
	3.2.1.2 Award Object .....	397
	3.2.1.3 AwardLineItem Object .....	399
	3.2.1.4 AwardPurchaseType Object .....	401
	3.2.1.5 BCASAward Object .....	402
	3.2.1.6 Buyer Object .....	403
	3.2.1.7 BuyerAssignment Object .....	403
	3.2.1.8 BuyerNote Object .....	404
	3.2.1.9 CancellationCode Object .....	406
	3.2.1.10 Clause .....	407
	3.2.1.11 ClauseCertification .....	408
	3.2.1.12 Communicator Object .....	409
	3.2.1.13 CompetitionCode Object .....	410
	3.2.1.14 Contact Object .....	411
	3.2.1.15 ControlStandards Object .....	412
	3.2.1.16 ControlVersion Object .....	412
	3.2.1.17 Currency Object .....	412
	3.2.1.18 DeliverySchedule Object .....	413
	3.2.1.20 DocumentAddressee Object .....	415
	3.2.1.23 DocumentType Object .....	416
	3.2.1.24 DocumentVersion Object .....	417
	3.2.1.25 DocumentVersionType Object .....	418
	3.2.1.26 DownloadStockClass Object .....	419



---

---

## Table of Contents

---

3.2.1.27 FOBAcceptancePoint Object .....	419
3.2.1.28 FederalStockClass Object .....	420
3.2.1.29 FreeOnBoard Object .....	421
3.2.1.30 FunctionalAck Object .....	421
3.2.1.31 FunctionalGroupHdr Object .....	422
3.2.1.32 FundCode Object .....	422
3.2.1.33 GSDefaults Object .....	423
3.2.1.34 HoldStatus Object .....	424
3.2.1.35 Holidays Object.....	425
3.2.1.36 ISAAuthQualifier Object .....	426
3.2.1.37 ISADefaults Object .....	426
3.2.1.38 ISAInterchangeQualifier Object .....	427
3.2.1.39 InterchangeControlHdr Object.....	428
3.2.1.41 InvoiceAddress Object .....	429
3.2.1.42 Item Object.....	430
3.2.1.43 ItemDetails Object .....	431
3.2.1.44 ItemPackageType Object .....	432
3.2.1.45 ItemWeightType Object.....	433
3.2.1.46 LineItem Object .....	434
3.2.1.47 LineItemStatus Object.....	435
3.2.1.48 Marks Object.....	436
3.2.1.49 MarksQualifier Object .....	437
3.2.1.50 MeasurementApplicationCode Object .....	438
3.2.1.51 MeasurementData Object.....	439
3.2.1.52 Message Object .....	440
3.2.1.53 MessageFrom Object .....	441
3.2.1.54 MessageReference Object .....	442
3.2.1.55 MessageTextBody Object .....	443
3.2.1.56 MessageTo Object.....	444
3.2.1.57 NegotiationAuthority .....	445
3.2.1.58 Nomenclature Object .....	446
3.2.1.59 Note Object .....	447
3.2.1.60 NoteStatus Object .....	448
3.2.1.61 OpenPurchaseRequest Object .....	449
3.2.1.62 Opr Object.....	450
3.2.1.63 OrganizationalEntity Object .....	451
3.2.1.64 OriginalTransaction Object .....	452
3.2.1.65 PTCType Object .....	453
3.2.1.66 Packaging Object .....	454
3.2.1.67 PaperworkType Object .....	455
3.2.1.68 Part Object.....	456
3.2.1.69 PartIdentifier Object.....	457
3.2.1.71 PkgCharacteristicCode Object .....	459

---

---

## Table of Contents (Continued)

---

3.2.1.72 PkgDescriptionCode Object.....	460
3.2.1.73 PolicyTermsAndConditions Object .....	461
3.2.1.74 PreferredAccess Object.....	462
3.2.1.75 PriorityGroup Object.....	463
3.2.1.76 Project Object.....	464
3.2.1.77 PurchaseOrderAck Object.....	465
3.2.1.78 PurchaseOrderChangeAckReq Object.....	466
3.2.1.79 Quote Object .....	467
3.2.1.80 QuoteLineItem Object.....	469
3.2.1.81 QuoteTerms Object .....	471
3.2.1.82 QuoteTypeCode Object.....	472
3.2.1.83 RedirectReason Object.....	473
3.2.1.84 RelatedPaperwork Object .....	474
3.2.1.85 ReqForQuote Object .....	475
3.2.1.86 ReqForQuoteLineItem Object.....	476
3.2.1.87 RequiredResponseTime Object.....	478
3.2.1.88 ReviewStatus Object .....	479
3.2.1.89 SADBUs Object .....	480
3.2.1.90 SendTo Object.....	481
3.2.1.91 Ship Object.....	482
3.2.1.92 ShippingDeliveryTypes Object.....	483
3.2.1.93 ShippingDocPackage Object.....	484
3.2.1.94 ShippingDocTypes Object .....	485
3.2.1.95 Signal Object.....	486
3.2.1.96 SolicitationHistory Object.....	487
3.2.1.97 SolicitationLineItem Object.....	488
3.2.1.98 SolicitationLineItemError Object .....	489
3.2.1.100 TechnicalErrorDescription Object .....	491
3.2.1.101 TermsBasis Object .....	492
3.2.1.102 TermsMethods Object.....	493
3.2.1.103 Text Object.....	494
3.2.1.104 TransactionReference Object.....	495
3.2.1.105 TransactionSent Object .....	496
3.2.1.106 TypeOfMeasurement Object.....	497
3.2.1.107 Unit Object.....	498
3.2.1.108 UnitOfMeasure Object.....	499
3.2.1.109 UnitPriceCodeBasis Object.....	500
3.2.1.110 UserManagerDefaults Object.....	501
3.2.1.111 adrs Object .....	502
3.2.1.112 VariationType Object.....	503
3.2.1.113 Variations Object .....	504
3.2.1.114 Vendor Object .....	505
3.2.1.115 VendorAward Object .....	506

---

---

## Table of Contents (Continued)

---

	3.2.1.116 VendorContact Object.....	507
	3.2.1.117 VendorHistory Object .....	508
	3.2.1.118 VendorQuoteLineItem Object.....	509
3.2.2	The GATEC Database Schema .....	510
3.2.3	Detailed Schema Description .....	510
	3.2.3.1 ACCTG Table .....	511
	3.2.3.2 ACQUISITION Table .....	512
	3.2.3.3 ACTIVESTATUS Table .....	513
	3.2.3.4 AWARD Table .....	514
	3.2.3.5 AWARDLINEITEM Table.....	515
	3.2.3.6 AWARDPURCHASETYPE Table.....	516
	3.2.3.7 BCASAWARD Table .....	517
	3.2.3.8 BUYER Table .....	518
	3.2.3.9 CANCELLATIONCODE Table .....	519
	3.2.3.10 CLAUSE Table .....	520
	3.2.3.11COMMUNICATOR Table .....	521
	3.2.3.12 CONTACT Table.....	522
	3.2.3.13 CONTROLSTANDARDS Table.....	523
	3.2.3.14 CONTROLVERSION Table .....	524
	3.2.3.15 DOCUMENT Table .....	525
	3.2.3.16 DOCUMENTADDRESSEE Table .....	526
	3.2.3.17 DOCUMENTSENT Table .....	527
	3.2.3.18 DOCUMENTSTATUS Table .....	528
	3.2.3.19 DOCUMENTTYPE Table .....	529
	3.2.3.20 DOCUMENTVERSION Table.....	530
	3.2.3.21DOCUMENTVERSIONTYPE Table.....	531
	3.2.3.22 FREEONBOARD Table .....	532
	3.2.3.23 FSCSIC Table .....	533
	3.2.3.24 FUNCTIONALGROUHDR Table .....	534
	3.2.3.25 GSDEFAULTS Table .....	535
	3.2.3.26 HOLDSTATUS Table .....	536
	3.2.3.27 HOLIDAYS Table .....	537
	3.2.3.28 INTERCHANGECONTROLHDR Table.....	538
	3.2.3.29 ISADEFAULTS Table.....	539
	3.2.3.31 LINEITEM Table.....	541
	3.2.3.32 MEASUREMENTAPPLICATIONCODE Table .....	542
	3.2.3.33 MEASUREMENTDATA Table .....	543
	3.2.3.34 MESSAGE Table .....	544
	3.2.3.35 MESSAGEFROM Table.....	545
	3.2.3.36 MESSAGEREFERENCE Table .....	546
	3.2.3.37 MESSAGETEXTBODY Table .....	547
	3.2.3.38 MESSAGETO Table.....	548

---

---

## Table of Contents (Continued)

---

3.2.3.39	NOMENCLATURE Table .....	549
3.2.3.40	NOTE Table .....	549
3.2.3.41	OFFLINERFQS Table .....	551
3.2.3.42	OPR Table .....	552
3.2.3.43	ORIGINALTRANSACTION Table .....	553
3.2.3.44	PACKAGING Table .....	554
3.2.3.45	PART Table .....	555
3.2.3.46	PIINS Table .....	556
3.2.3.47	PREOPR Table .....	557
3.2.3.48	PRIORITYGROUP Table .....	558
3.2.3.49	QUOTE Table .....	559
3.2.3.50	QUOTELINEITEM Table .....	560
3.2.3.51	QUOTETERMS Table .....	562
3.2.3.52	RELATEDPAPERWORK Table .....	563
3.2.3.53	REQFORQUOTE Table .....	564
3.2.3.54	REQFORQUOTELINEITEM Table .....	565
3.2.3.55	REVIEWSTATUS Table .....	567
3.2.3.56	SADBU Table .....	568
3.2.3.57	SHIP Table .....	569
3.2.3.58	SHIPPINGDOCPACKAGE Table .....	570
3.2.3.59	ITECONFIGURATION Table .....	571
3.2.3.60	SOLICITATIONHISTORY Table .....	572
3.2.3.61	SOLICITATIONLINEITEM Table .....	573
3.2.3.62	SOLICITATIONLINEITEMERROR Table .....	574
3.2.3.63	STATUSOPERATION Table .....	575
3.2.3.64	STMNT Table .....	576
3.2.3.65	TECHNICALERRORDESCRIPTION Table .....	577
3.2.3.66	TERMSBASIS Table .....	578
3.2.3.67	TEXT Table .....	579
3.2.3.68	TRANSACTIONSENT Table .....	580
3.2.3.69	UNIT Table .....	581
3.2.3.70	UNITOFMEASURE Table .....	582
3.2.3.71	USERMANAGERDEFAULTS Table .....	583
3.2.3.72	VADRS Table .....	584
3.2.3.73	VARIATIONS Table .....	585
3.2.3.74	VENDOR Table .....	586
3.2.4	NARQ Code Generation Utility .....	587
3.2.5	NARQDEF Detail Reference .....	587
3.2.5.1	DATATYPE Table .....	587
3.2.5.2	DERIVEDOBJECT Table .....	588
3.2.5.3	OBJECT Table .....	589

---

---

## Table of Contents (Continued)

---

	3.2.5.4 OBJECTCONSTANTS Table .....	590
	3.2.5.5 OBJECTELEMENT Table .....	591
	3.2.5.6 OBJECTRELATIONSHIP Table .....	592
	3.2.5.7 RELATION Table.....	593
	3.2.5.8 SIMPLEOBJECT Table.....	594
3.3	Development Environment .....	595
	3.3.1 Building Libraries .....	596
	3.3.2 Making Changes to Libraries .....	597
3.4	Database Connection.....	598
	3.4.1 Searching a Single Table.....	600
	3.4.2 Creating a New Record .....	603
	3.4.3 Deleting an Existing Record .....	605
3.5	Glossary of Database Terms .....	607
 SECTION 4	 CDFDB Library .....	 609
4.1	Design Intent .....	609
4.2	Dependencies .....	609
4.3	Advantages to the CDF Approach .....	613
4.4	Disadvantages to the CDF Approach.....	614
4.5	Short Comings (Implementation).....	614
4.6	CDFtoDB .....	614
4.7	Interface Description (chk_mand) .....	616
4.8	Creating New Applications for New Document Types .....	618
4.9	Existing Applications .....	620
	4.9.1 843CDFtoDB .....	620
	4.9.2 838cCDFtoDB .....	620
	4.9.3 824CDFtoDB .....	620
	4.9.4 864CDFtoDB .....	621
4.10	Short Comings.....	621
4.11	DBtoCDF .....	622
4.12	Interface Description for DBtoCDF() .....	624
4.13	Creating routines for new document types .....	625
	4.13.1 Applications using DBtoCDF .....	626
	4.13.1.1 840DBtoCDF .....	627
	4.13.1.2 850DBtoCDF .....	627
	4.13.1.3 BCASDBtoCDF .....	628
	4.13.2 DBtoCDF Short Comings .....	628
4.14	Building and Testing .....	628
4.15	CDFDB Unit Testing .....	629
4.16	System Testing .....	629
4.17	System Install.....	629

---

---

---

## Table of Contents (Continued)

---

4.18	Diagnostic Error Messages .....	629
SECTION 5	Transport .....	631
5.1	Transport Overview .....	631
5.2	Transport Approach .....	632
5.3	Addressing .....	633
5.4	Outbound .....	635
5.5	Inbound .....	636
5.6	Transport Support Software .....	639
	5.6.1 input .....	639
	5.6.2 newsyslog .....	640
	5.6.3 cdfretry .....	641
5.7	Future Enhancements .....	641
5.8	Configuration Dependencies .....	642
SECTION 6	GATEC 2 Test Matrix .....	643
6.1	The Matrix .....	643

---

## SECTION 1 Interface to Legacy System

---

The software that comprises the interface to the legacy system is primarily located at `$CVSROOT/src/wang` in the GATEC development environment. Other support software is distributed in `$CVSROOT/tisp`, `$CVSROOT/db`, `$CVSROOT/que`, and `CVSROOT/narqdb/src/bin/readopr2`.

---

### 1.1 BCAS Download of RFQs

---

#### NAME

*getopr\_bsp\_cron3* - get new open purchase requests (and associated item description) from the Wang BCAS system and load them into the GATEC database.

#### SYNOPSIS

`getopr_bsp_cron3 [ -s ]`

#### DESCRIPTION

*getopr\_bsp\_cron3* downloads all new BCAS open purchase requests that match the system-wide download criteria. It checks for the existence of the `$LOCKFILE`, `/home/bcas/getopr_bsp_cron_IS_RUNNING`. If `$LOCKFILE` exists, *getopr\_bsp\_cron3* assumes that an instance of itself is already running, else it creates `$LOCKFILE`.

*getopr\_bsp\_cron3* calls "`getwangfiles -p cpopr`" to obtain the file `OPR_ALL.dat` from the Wang. This file contains all current open purchase requests on the Wang BCAS system. `OPR_ALL.dat` is loaded into the `PreOPR` table by calling `sqlldr` to actually load into the SQL View `v_preopr`. `v_preopr` implements constraints on `OPR` records, and accepts those `OPR` records where:

1. The `OPR` does not already exist in the `Solicitation-LineItem` `rdbs` table.

2. The BSP (buyer id) in the OPR is in the DownloadBuyers rdbms table.

3. The total estimated price of the OPR is less than or equal to the value of the EstimatedPriceLimit column in the UserManagerDefaults rdbms table.

4. The priority of the OPR is greater than or equal to the value of the MaximumPriority column in the UserManagerDefaults rdbms table.

Once all the new records have been loaded into the PreOpr table, a list of stock numbers for which item and nomenclature descriptions are needed is generated from the PreOpr table.

*getopr\_bsp\_cron3* calls "getwangfiles -p cpitstk" with up to 14 of these stock number item descriptions. For each group of stock numbers, getwangfiles will obtain from the Wang two files: ITEM\_ALL.dat and NOME\_ALL.dat. These two files are loaded into the Item rdbms table and the Nomenclature rdbms table. Then, the rdbms table Opr is deleted and recreated, and those OPR's in the PreOpr table with associated Item records are loaded into the Opr table. Finally, the program *readopr2* will read all records from the Opr table, and place them into the appropriate rdbms tables such that buyers logged into GATEC will see the new OPR's on their "Unissued" screen.

*getopr\_bsp\_cron3* can only download 14 Item descriptions at a time, so it loops over each group of 14, calling *readopr2* for each group.

When *getopr\_bsp\_cron3* exits, it removes the \$LOCKFILE.

## INTERNAL DESCRIPTION

*getopr\_bsp\_cron3* gets new open purchase requests (and the associated item description) from the Wang BCAS system and loads them into the GATEC database.

It is run roughly once an hour during business hours, via the UNIX "cron" command.

## Control Flow

```
if lockfile exists
then
    exit (a previous invocation is still running)
```



```
else
    create lockfile
fi
```

set signal handling to remove lockfile on program termination

Log onto the Wang and run the Wang procedure "cpopr" to create a copy of the Wang OPR file.

Download the OPR file to the local UNIX computer via ftp.

```
if unsuccessful
then
    exit
fi
```

Drop and recreate the sql table "preopr"

Reload the preopr table with the OPR file we just downloaded, throwing away all records that don't match our download criteria.

Generate a list of all the unique stock numbers in the preopr file, ordered by frequently used stock numbers first. We'll need to obtain the item and nomenclature descriptions of these stock numbers from BCAS.

```
While [ more stock numbers on the list of unique stock numbers ]
do
```

```
    Pop up to 14 stock numbers from the unique list onto the
download list
```

Log onto the Wang and use the download list and the Wang procedure "cpitstk" to create two files on the Wang: one containing the item descriptions of the stock numbers, and one containing the nomenclature descriptions of the stock numbers.

```
    Download those two files to the local UNIX computer via
ftp.
```

```
        if unsuccessful
        then
            exit
        fi
```

```
        Drop and recreate the sql tables "item" and "nomenclature".
```

```
files
    Reload the item and nomenclature tables with the BCAS
we just downloaded.
```

Drop and recreate the sql table "opr".

Move all records from table "preopr" to table "opr" that have stock numbers in the newly loaded "item" table.

Invoke the external program "readopr2" to read the opr, item, and nomenclature tables, and insert new records into the rest of the database. Once those new records are inserted, buyers may begin examining them.

```
        if unsuccessful
        then
            exit
        fi
done
```

## OPTIONS

-s Skip the step of obtaining a new OPR file. This is mostly useful for debugging and database reloading purposes.

## DATABASE TABLES

v_preopr	view into which OPR_ALL.dat is inserted
PreOpr	table into which OPR_ALL.dat is loaded. Also defines the view v_preopr.
Item	table into which ITEM_ALL.dat is loaded
Nomenclature	table into which NOME_ALL.dat is loaded
DownloadBuyers	table containing valid GATEC buyers SolicitationLineItem table containing existing GATEC OPR's UserManagerDefaults table containing the maximum estimated price and the maximum priority of OPR's to download.

## FILES

\$LOCKFILE	lock file
\$HOME/set_ecedi_env	sets gateway env vars
OPR_ALL.dat	fixed-format file containing all BCAS OPR's
ITEM_ALL.dat	fixed-format file containing up to 14 BCAS Item records
NOME_ALL.dat	fixed-format file containing up to 14 BCAS Nomenclature records
Item2.ctf	Oracle control file for loading ITEM_ALL.dat
Nomenclature2.ctf	Oracle control file for loading NOME_ALL.dat
/usr/spool/cron/crontabs/gatecmgr	controlling cron file
/tmp/getopr_bsp_allstocknums\$\$	
/tmp/getopr_bsp_somestocknums\$\$	
/tmp/getopr_bsp_tmpstocknums\$\$	

tmp files containing stock numbers

#### SEE ALSO

*getwangfiles(1)*, *BCASrunproc(1)*, *downloadch(1)*,  
*readopr2(1)*, *sh\_get\_login\_info(1)*, *cpopr(WANG)*,  
*cpitstk(WANG)*, *sqlplus(1LOCAL)*, *sqlldr(1)*, *outline\_wade(1)*,  
*getopr\_bsp\_cron3.pdl*

#### BUGS

*getopr\_bsp\_cron3* can take from 5 minutes to several hours to run, depending on the number of new OPR's to download and the speed of the TCP/IP link from the GATEC host to the Wang.

There is a fair chance that *getopr\_bsp\_cron3* will prematurely terminate due to some Wang error. This is not a problem, because *getopr\_bsp\_cron3* will pick up where it left off the next time it runs.

The Wang is down for approximately 1 hour every day (Monday through Friday) starting at 11:30 AM local time.

#### NOTES

Through experimentation, we have found that an acceptable cron frequency is to run *getopr\_bsp\_cron3* about once an hour >from 0600 through 1800. From 1200 to 1300 we increase the frequency to once every ten minutes, so we can catch the Wang as soon as it comes back up. Example cron entry:

```
# Downloads new open purchase requests from Wang BCAS 30 6-10,13-18 * * 1-5
/home/gatec2/bin/getopr_bsp_cron3
# Do the lunch rush
10,20,30,40 12 * * 1-5 /home/gatec2/bin/getopr_bsp_cron3
```

---

### 1.1.2 Support Software for RFQ download

---

#### NAME

*readopr2* - insert WANG BCAS Open Purchase Requests into the database.

#### SYNOPSIS

*readopr2*

## DESCRIPTION

*readopr2* assumes that the Opr, Item, and Nomenclature tables contain new WANG BCAS Open Purchase Requests (OPRs). It also assumes that the Unit, Ship, and SiteConfiguration tables contain system information.

*readopr2* reads from these tables, and inserts rows into the ReqForQuote, LineItem, ReqForQuoteLineItem, Document Acquisition, SolicitationLineItem, and Part tables.

*readopr2* exits 0 on success, 1 on failure.

## DATABASE TABLES

Opr	BCAS OPRs.
Item	Item descriptions of the stock numbers in the Opr table.
Nomenclature	If the Item description contains more than six lines of description, the rest of the description is in this table.
Unit	Conversions between Unit Of Issue and Unit Of Measure.
Ship	Copy of the BCAS Ship file.
SiteConfiguration	Contains the Site Address (DODAAC) of the local contracting office.
ReqForQuote	
LineItem	
ReqForQuoteLineItem	
Document	
Acquisition	
SolicitationLineItem	
Part	

## SEE ALSO

*getopr\_bsp\_cron3(1)*, *Connection(3N)*, *Database(3N)*, *Table(3N)*, *get\_login\_info(?)*

---

## 1.2 Download of other BCAS specific Data

---

### NAME

*getstmntship\_cron* - get the Wang BCAS files stmnt, ship, vadr and acctg

## SYNOPSIS

getstmntship\_cron

## DESCRIPTION

getstmntship\_cron logs onto the Wang BCAS system, creates the files stmnt, ship, vadr and acctg, ftp's the files down to the GATEC site, and loads the files into the GATEC data base.

## INTERNAL DESCRIPTION

*getstmntship\_cron* first checks for the existence of the \$LOCKFILE, /home/bcas/getstmntship\_cron\_IS\_RUNNING. If \$LOCKFILE exists, *getstmntship\_cron* assumes that an instance of itself is already running, else it creates \$LOCKFILE. *getstmntship\_cron* calls "getwangfiles -p cpbcas" to run the Wang procedure cpbcas, and download via ftp all files created by the Wang procedure.

*getstmntship\_cron* drops and recreates the GATEC rdbms tables "Ship", "Stmnt", "Acctg", and "Vadr" with separate sqlplus commands. Then, using separate sqlldr commands, it loads each downloaded file into the GATEC rdbms.

## DATABASE TABLES

Stmnt	table into which stmnt.dat is loaded.
Ship	table into which ship.dat is loaded.
Acctg	table into which acctg.dat is loaded.
Vadr	table into which vadr.dat is loaded.

## FILES

\$LOCKFILE	lock file
\$HOME/set_ecedi_env	sets gateway env vars
stmnt.dat	fixed-format file containing all BCAS Order Statements.
Stmnt.ctl	Oracle control file for loading stmnt.dat.
ship.dat	fixed-format file containing all BCAS Ship-to data.
Ship.ctl	Oracle control file for loading ship.dat.
acctg.dat	fixed-format file containing all BCAS Accounting data.
Acctg.ctl	Oracle control file for loading acctg.dat.
vadr.dat	fixed-format file containing all BCAS Vendor Address data.

Vadrs.ctl

Oracle control file for vadrs.dat.

## SEE ALSO

*getwangfiles(1)*, *BCASrunproc(1)*, *downloadch(1)*,  
*cpbcas(WANG)*, *sqlplus(1LOCAL)*, *sqlldr(1)*, *ORACLE RDBMS*  
*Utilities User's Guide*

## BUGS

The vadrs file (Vendor address) is large, about 4 megabytes. Successful ftp from the wang is sometimes difficult. Keep trying.

If any active user on the system (e.g., a GATEC buyer) access one of the above DATABASE TABLES after it has been deleted, but before it has been reloaded, results are undefined. It is recommended that this procedure be run by the system manager while the system is quiescent. *getstmntship\_cron* should really handle this gracefully.

---

## 1.3 Utilities Used for Download of BCAS Data

---

### NAME

*getwangfiles* - run a procedure on the Wang and get the files created by the procedure

### SYNOPSIS

*getwangfiles* -p cpop | cpbcas

cat <fileofstocknumbers> | *getwangfiles* -p cpitstk

### DESCRIPTION

*getwangfiles* invokes BCASrunproc(1) to log onto the Wang VS system and run either the Wang procedure cpop(WANG), cpbcas(WANG), cpitstk(WANG).

BCASrunproc(1) creates one or more files on the Wang. *getwangfiles* uses downloadch(1) to obtain those files via FTP and place them in the current working directory.

*getwangfiles* exits 0 on success, 1 on failure.

## OPTIONS

-p cpopr | cpbcas | cpitstk procedure to run on the wang.

## LIMITATIONS

The Wang host domain name and the Wang userid for login are hard coded in the program. The current values are

download\_host='wpwan08.wpafb.af.mil'

download\_user='g2w'

## FILES

\$LOCKFILE	lock file
\$HOME/set_ecedi_env	sets gateway env vars
/usr/spool/cron/crontabs/gatecmgr	controlling cron file
/tmp/getwangdat\$\$	tmp file containing data to be passed to BCASrunproc
/tmp/getwanggwout\$\$	tmp file containing BCASrunproc output

## SEE ALSO

*BCASrunproc(1)*, *downloadch(1)*, *cpopr(WANG)*, *cpitstk(WANG)*, *getwangfiles.pdl*

## NAME

*BCASrunproc* - Gateway program to run a procedure on a WANG VS system.

## SYNOPSIS

*BCASrunproc* -l BCASaccount [ -h ] [ -p password-fR ] [ -s BCASsystem ] [ -t timeout ] cpopr | cpbcas

*BCASrunproc* -l BCASaccount -f fileofstocknumbers [ -h ] [ -p password-fR ] [ -s BCASsystem ] [ -t timeout ] cpitstk

## DESCRIPTION

*BCASrunproc* uses telnet to connect to BCASsystem and log on using BCASaccount. It emulates a human user logged in to the Wang. From the Wang Command Processor menu, it invokes one of the procedures cpopr, bpcas, or cpitstk. It assumes procedure is in volume VOL333 and library G2WPGM. The procedures cpopr

and cpbcas expect no input from *BCASrunproc*. The procedure cpitstk expects a series of up to 14 stock numbers to be input.

*BCASrunproc* greps for all Wang output of the form "file ... created", and echoes that output to stdout. When the procedure finishes and returns to the Command Processor menu, *BCASrunproc* then logs out from the Wang.

*BCASrunproc* exits 0 on success, 1 on failure.

## INTERNAL DESCRIPTION

Set up interrupt handling

call procedure cmdargs to parse the command line, obtain:

BCASaccount, BCASpassword(deprecated), BCASsystem, \_timeout, program, and perhaps stock\_file.

connect to bcas via telnet.

call procedure bcaslogin

if ! ok

then

exit

fi

Output pfkey(1) to get to "Run program or procedure" menu

if didn't get there

then

exit

fi

Output program to run

Look for bcas response "Procedure ... in progress"

if ! found

then

exit

fi

while [1] # loop forever

do

if no response from wang in 120 seconds

then

break

fi

if response was "Procedure .. in progress"

then

continue

fi



```

if response was "procedure ... beginning"
then
    continue
fi
if response was "file ... created"
then
    continue
fi
if response was "procedure ... finished"
then
    continue
fi
if response was "Wang VS Command Processor"
then
    break
fi
if response was "GETPARM ... Correction Required"
then
    break
fi
if response was "Enter stock number"
then
    read a stock number from the stock number file on
cmd line,
    and output it.
    continue
done
logout from bcas

```

## OPTIONS

-l BCASaccount	Wang BCAS userid to use.
-f fileofstocknumbers	file containing up to 14 stock numbers, one per line.
-h	print usage message.
-p password	Wang BCAS password to use. Specifying this option creates a security hazard, because a utility such as ps(1) can obtain the <i>BCASrunproc</i> command-line arguments, hence the password.
-s BCASsystem	TCP/IP domain name of Wang. If not specified, defaults to wpwan08.wpafb.af.mil.

-t timeout                      Set timeout period for term statements, in seconds. If not specified, defaults to 90 seconds.

## ENVIRONMENT

\$ECLIB must be set. See FILES.

## FILES

\$ECLIB/gatewaylog	File where Gateway puts log messages.
\$ECLIB/coredir	Directory where Gateway puts core dumps.
/etc/.authlist	File containing encrypted password for the specified BCASaccount. This file is NOT be world-readable.

## BUGS

If this program is running without a controlling terminal, (e.g., if it is invoked from cron), then it will not normally write anything to stdout. A work-around is to have cron (or the process spawned by cron) invoke it like this:

```
echo "" | BCASrunproc
```

## SEE ALSO

*BCASupload(1), bcasprocs\_m4(3), BCASrunproc\_m4.pdl,*

---

## 1.4            BCAS Award Upload

---

### NAME

*putuploads\_cron* - make awards on BCAS

### SYNOPSIS

*putuploads\_cron* BCASaccount

### DESCRIPTION

*putuploads\_cron* examines GATEC's bcasupload queue to determine if there are any awards on the queue. If so, it logs onto the Wang BCAS system using the Wang BCAS account

BCASaccount and makes one award for each item on the queue. If the award is successful, *putuploads\_cron* passes the award data to the process that translates the award data into an ANSI X12 850 transaction. If the award fails due to BCAS being down, *putuploads\_cron* re-queues the award data onto the bcasupload queue and exits. If the award fails due to an error in the award data, *putuploads\_cron* will send the award back to the originating buyer, and will continue with the next award.

## INTERNAL DESCRIPTION

*putuploads\_cron* checks for the existence of the \$LOCKFILE, /home/bcas/*putuploads\_cron* \_IS\_RUNNING<BCASaccount>. If \$LOCKFILE exists, *putuploads\_cron* assumes that an instance of itself is already running, else it creates \$LOCKFILE. *putuploads\_cron* checks to see that the bcasupload queue is "UP". If so, it pops the next item off the queue. A queue item consists of two things. The first, the "Key", is the file containing Wang BCAS award data. The second, the "Data", is one or more filenames containing associated award information that will be translated into ANSI X12 transactions.

*putuploads\_cron* verifies that the "Key" file is correctly formatted by first filtering the file through seluploadcdf, then checking the result with cdf\_check. Errors here cause two things to happen:

1. The GATEC award document is put back on the buyer's "Closed" pile, with the program setUTN\_aw\_to\_cl.
2. The text of the error message is sent to the buyer with the program acqerr.

Assuming there are no errors with the file, it is sent to the program BCASupload. If BCASupload reports:

"upload succeeded" then the award has been made on BCAS, and each file in the "Data" is then sent for outbound processing with the command:

lpr -Poutbound \$File

If BCASupload reports an error of the form:

BCAS reports .\* error then the text of the error message is sent to the buyer with the program acqerr.

If there are any other errors, there was an abnormal occurrence somewhere between the GATEC upload application BCASupload and the Wang BCAS program that processes awards. Abnormal

occurrences cause the award to be re-queued, and *putuploads\_cron* exits.

When *putuploads\_cron* exits, it removes the \$LOCKFILE.

## INTERNAL DESCRIPTION

### Control Flow

If lockfile exists

then

exit (a previous invocation is still running)

else

create lockfile

fi

Set signal handling to remove lockfile on program termination

If the "bcasupload" queue is not UP

then

exit

fi

While [ there are more items on the "bcasupload" queue ]

do

Pop the item off the queue. An item consists of two files:

1. an upload file used to make an award on BCAS
2. a complex second file containing data used to generate an X12 850.

If the upload file contains any syntax errors

then

Set the state of the acquisition from "Awarded" to "Closed"

Send an error message to the responsible buyer.

Continue to end of while loop to get next item.

fi

Run the local program "BCASupload" to log onto the Wang, emulate a buyer sitting at a terminal, and make an award.

If "BCASupload" reports "upload succeeded"

then

Send the complex file (via lpr) for further processing.

Continue to end of while loop to get next item.

fi

If "BCASupload" reports an error that the buyer can fix,

```

        e.g., "No Such Vendor ID", "Invalid Negotiation
        Authority", etc...
    then
        Set the state of the acquisition from "Awarded" to
        "Closed"
        Send an error message to the responsible buyer.
        Continue to end of while loop to get next item.
    else if "BCASupload" reports some other kind of error, e.g.,
        it couldn't connect to the Wang because the Wang is
        down,
    then
        Re-queue the data onto the bcasupload queue
        exit by breaking out of the while loop.
        (Assume that when cron runs this program in a half
        hour, bcas will be back up)
    fi
done (while loop)

```

## FILES

\$LOCKFILE	lock file
\$HOME/set_ecedi_env	sets gateway env vars
/usr/spool/cron/crontabs/gatecmgr	controlling cron file
/tmp/upload_error\$\$ t	tmp file containing upload errors
/tmp/uploadcdf\$\$	tmp file containing name-value pairs to be passed to BCASupload
/tmp/BCASuploadout\$\$	tmp file containing output of BCASupload

## SEE ALSO

*BCASupload(1), acqerr(1), setUTN\_aw\_to\_cl(1), get\_UTNNumber\_from\_cdf(1), get\_piin\_from\_cdf(1), seluploadcdf(1), cdf\_check(1), q(3), errtomgr(1), lpr(1), putuploads\_cron.pdl*

## NOTES

Through experimentation, we have found that an acceptable cron frequency is once every half hour during business hours. Sample cron entry:

```

# Perform uploads of new awards to wang BCAS
25,55 6-10,12-19 * * 1-5 /home/gatec2/bin/putuploads_cron

```

## NAME

*BCASupload* - Gateway program to make an award on the Wang BCAS system

## SYNOPSIS

BCASupload BCASaccount [ -h ] [ -p password-fR ] [ -s BCASsystem ] [ -t timeout ] cdffile

## DESCRIPTION

*BCASupload* logs into BCAS over TCP and walks through the menus to get to the AWARD menu. *BCASupload* uses the values in cdffile to make an award. If the %contract\_number in cdffile exists, *BCASupload* performs a Delivery Order award. If %contract\_number is blank or non-existent, *BCASupload* performs a Purchase Order award.

*BCASupload* exits 0 on success, something else on failure.

## INTERNAL DESCRIPTION

Define all strings used in the cdf  
Set up (some) interrupt handling

call procedure cmdargs to parse the command line, obtain:

BCASaccount, BCASpassword(deprecated), BCASsystem, \_timeout, program, and CDFfilelist.

connect to bcas via telnet.

call procedure bcaslogin

if ! ok

then

exit

fi

Make sure we're at the System Administrator's screen, if didn't get there

then

exit

fi

call procedure "walkmenus" to get to the AWARD menu.

if didn't get there

then

exit

fi

Load cdf file into internal gateway variables

Set all possible bcas error messages

```
if this is a normal PO (no contract_number)
then
    output the pf1 key to get to the PRICED PURCHASE
    ORDER AWARD screen
else
    output the pf2 key to get to the DELIVERY ORDER
    AWARD PROCESS screen
fi
if we didn't get there
then
    exit
fi
```

```
call procedure "upload" to fill the data into the BCAS forms.
if ! ok
then
    exit
fi
```

call procedure "backout" to log off from BCAS

## OPTIONS

-l BCASaccount	Wang BCAS userid to use.
-h	print usage message.
-p password	Wang BCAS password to use. Specifying this option creates a security hazard, because a utility such as ps(1) can obtain the BCASrunproc command-line arguments, hence the password.
-s BCASsystem	TCP/IP domain name of Wang. If not specified, defaults to wpwan08.wpafb.af.mil.
-t timeout	Set timeout period for Gateway term statements, in seconds. If not specified, defaults to 90 seconds.

## ENVIRONMENT

\$ECLIB        must be set. See FILES.

## FILES

\$ECLIB/gatewaylog	File where Gateway puts log messages.
\$ECLIB/coredir	Directory where Gateway puts core dumps.
/etc/.authlist	File containing encrypted password for the specified BCASaccount. This file should NOT be world-readable. Recommend chmod 600 /etc/.authlist.

## BUGS

If this program is running without a controlling terminal, (e.g., if it is invoked from cron), then it will not normally write anything to stdout. A work-around is to have cron (or the process spawned by cron) invoke it like this:

```
echo "" | BCASrunproc
```

## SEE ALSO

*bcasupl\_m4.pdl*, *bcasprocs\_m4(3)*, *bcasupl\_m4(3)*, *Gateway Programmer's Guide [REF000]*

## BCAS Award Upload CDF File Format

An example of an award CDF file is shown below,

```
%Xbegin
%Xpurpose   BCAS award
%Xfilename  BCASCDF
%Xdestination_host  BCAS
%Xversion   2
%Xdate      94 01 11
#***** start of data elements *****
%award_date 94JAN11
%award_piin 94EF847
%buyer_code G1H
%another_fed_agency      N
%competition_code  Y
%confirm_with
%contract_number
%contractor_signs  N
%discount_days_net 30
%do_rating  c9e
%fob_code  D
%line_item  0001
%negotiation_authority      0301
%number_of_line_items      1
%order_statements  EX IN SI GU
%purchase_variation
%quantity  00006
```



%required\_delivery\_date 94FEB11  
%review\_accounting\_class N  
%solicitation\_number 93R9011  
%special\_contract\_order\_preparation N  
%supplemental\_description N  
%unit\_price 1.0000  
%variation\_percent  
%vendor\_bcas\_code TMPT007  
%warranty\_clause\_days  
%UTNNumber F0000093R9011001  
%Xend

NAME

*setUTN\_aw\_to\_cl* - change the state of an Acquisition from Awarded to Closed.

SYNOPSIS

setUTN\_aw\_to\_cl UTNNumber [ AwardPIIN ]

DESCRIPTION

*setUTN\_aw\_to\_cl* changes an acquisition with the UTN Number UTNNumber and with a ReviewStatus of 'AW' to one with a ReviewStatus of 'CL'. This is usually done when an upload has failed for a reason that can be corrected by a buyer. Moving the Acquisition to 'CL' means that the buyer can then edit it, and re-award it.

*setUTN\_aw\_to\_cl* will also re-use AwardPIIN if it is specified. In general, AwardPIIN is always specified, but if the upload has failed due to the error 'Award Piin Already Used', then don't specify it.

TABLES

Acquisition	Table containing a UTNNumber's status.
Piins	Table containing AwardPIIN to reuse.
Award	
AwardLineItem	
BCASAward	

SEE ALSO

*Oracle PL/SQL User's Guide and Reference, [REF001], sh\_get\_login\_info(1)*

NAME

*get\_UTNNumber\_from\_cdf* - extracts a UTNNumber from stdin and displays it to stdout.

SYNOPSIS

`get_UTNNumber_from_cdf`

#### DESCRIPTION

*get\_UTNNumber\_from\_cdf* reads from stdin and prints the first occurrence of UTNNumber if a line of the form:

%UTNNumber UTNNumber

exists in stdin.

#### SEE ALSO

*putuploads\_cron(1)*

#### NAME

*get\_piin\_from\_cdf* - extracts an award piin from stdin and displays it to stdout.

#### SYNOPSIS

`get_piin_from_cdf`

#### DESCRIPTION

*get\_piin\_from\_cdf* reads from stdin and prints the first occurrence of piin if a line of the form

%award\_piin piin

exists in stdin.

#### SEE ALSO

*putuploads\_cron(1)*

#### NAME

*acqerr* - associate error text with a GATEC acquisition

#### SYNOPSIS

`acqerr -u UTNNumber [ -s subject ]`

## DESCRIPTION

*acqerr* reads from stdin and inserts text into the Oracle database. Such text may then be reviewed by the GATEC buyer that is handling the specified UTNNumber. *acqerr* exits 1 if there are any errors, else exists 0.

## DATABASE TABLES

SolicitationLineItemError Text	An instance of some error text. Multi-line text associated with one SolicitationLineItemError row.
-----------------------------------	--

## SEE ALSO

get\_login\_info(3), Programmer's Guide to the ORACLE Precompilers [REF002].

## BUGS

The SolicitationLineItemError table should probably be keyed on UTNNumber rather than on (SolicitationNumber, LineItem). As it is, UTNNumber must be considered a "smart key", containing SiteCode, SolicitationNumber, and LineItem.

---

### 1.4.3 Gateway Utilities Used in Upload of Award Data

---

## NAME

*resp\_err*, *back2AWD01*, *backout*, *checkvars*, *dumpvars*,  
*menumove*, *parsefields*, *officetosysadm*, *upload*, *walkmenus* -  
Award-specific WANG BCAS Gateway functions

## SYNOPSIS

call resp\_err(resp, var\_name)

call back2AWD01

call backout

call checkvars

call dumpvars

call menumove(pfk, menuname)

call parsefields(cdffile)

call upload

call walkmenus

## DESCRIPTION

*resp\_err* takes two arguments: *resp*, the response received >from the Wang, and *var\_name*, the error type, and displays an error message to stdout.

*back2AWD01* issues 4 calls to *pfkey*(1). Then it issues a *pfkey*("HELP"), followed by a *pfkey*(1), then calls *menumove* to move to the AWD01 screen.

*backout* moves the program from the Purchase Order or Delivery Order menu back to the System Administrator's screen, then calls *bcaslogout*.

*checkvars* checks that most needed variables are set, prior to performing an award. If all critical variables are set, the global variable *ok* is set to TRUE. Otherwise, an error message is printed to stdout.

*dumpvars* displays a number or critical variables to stdout.

*menumove* takes two arguments: *pfk*, a *pfk* (1-32) to send, and *menuname*, a menu to look for. If the menu is found, the global variable *ok* is set to TRUE, else it is set to FALSE. If *pfk* is 0, no stimulus is issued.

*parsefields* reads the file *cdffile*, which is assumed to be a cdf file with lines of the form

%name value.

For each line in the cdf, *parsefields* assign value to the global variable %name.

*upload* puts the data extracted from the CDF file into the award process menus. In other words, it performs the award. On success, the global variable *ok* is set to TRUE, otherwise *ok* is set to FALSE.

*walkmenus* routine handles the traversal of the menus to the AWARD menu. It is presumed that BCAS is on the SYSTEMS ADMINISTRATORS menu when this routine is called. On

success, the global variable ok is set to TRUE, otherwise ok is set to FALSE.

SEE ALSO

*BCASrunproc(1)*, *BCASupload(1)*

---

## 1.5 BCAS Item Description Upload

---

### NAME

*upload\_bcas\_item\_desc\_cron*

### SYNOPSIS

`upload_bcas_item_desc_cron`

### DESCRIPTION

Any item description, part number, and manufacturer information that is modified by a buyer before issuing an RFQ is uploaded back to BCAS via the Bourne Shell script *upload\_bcas\_item\_desc\_cron*. When an RFQ is issued, the GATEC application writes a CDF file containing the modified item, part, and manufacture data and places the CDF filename on the *bcasitem* queue. *upload\_bcas\_item\_desc\_cron* takes item upload CDF filenames off this queue (using the *qpop* utility), insures the data in them is correct (using *selitemcdf* and *cdf\_check*), then passes the filename to the gateway script *BCASitemupload* *BCASitemupload*, then uploads the information to the BCAS system.

### INTERNAL DESCRIPTION

*upload\_bcas\_item\_desc\_cron* makes use of the gateway script *BCASitemupload* to upload the content of item description CDF files (whose filenames reside on the *bcasitem* queue) back to BCAS.

After setting up file aliases and insuring that another version of itself is not running, *upload\_bcas\_item\_desc\_cron* sets up its own environment, then makes use of *qstatus* to insure the *bcasitem* item queue still exists. Next, *qstatus* is used to make sure the queue is operational. At this point the main loop is entered where the names of CDF files to be processed are repeatedly popped off the *bcasitem* queue via the *qpop* utility. After a check of the integrity

of the information popped off the queue (i.e. valid key and data), the existence of the CDF file is confirmed. Next, the Perl scripts *selitemcdf* and *cdf\_check* are used to check to syntax of specified fields in the file. If the syntax checks are passed, *BCASitemupload* is called with the name of the CDF file whose data is to be uploaded to BCAS. After execution a grep is made of *BCASitemupload* output. If the keywords "item upload succeeded" are detected, the name of the next file to process is popped off the queue for processing. If the success string is not found, then a check is made to determine if more than three consecutive errors have occurred, if so the cron script is stopped. If the error count is less than three, a check is made to determine whether *BCASitemupload* has recommended requeuing the file. If so (and it has not been requeued once), the file is requeued, otherwise a check is made to see whether the BCAS failure was because of a disconnect. If this was the case the file is requeued and the next file is processed--otherwise the troubled file will not be requeued and the next file will be examined by popping the bcascancel queue.

Processing (for non error situations) continues until the bcascancel queue is empty. Bourne Shell scripts *itemerrortouser*, and *errtomgr* are used to report errors to the gatec manager.

## TESTING

Developing a test fixture for this software is very straightforward. The following error conditions are detected:

lockfile exists (version of *upload\_bcas\_item\_desc\_cron* already running)

cannot touch lockfile

cannot touch summary file

*qstatus* unknown exit error

queue not up

queue not empty

cdf filename null

cdf file does not exist

grammar error in CDF file

unable to send 850 CDF out to outbound queue

more than three consecutive errors detected in upload

REQUEUE message from *BCASitemupload* detected

DISCONNECT message from *BCASitemupload* detected

failed requeue

To make sure each condition is handled correctly, each error is made to occur, then the output results can be verified.

## FILES

/home/bcas/\$LOCKFILE	Lock file
\$HOME/set_ecedi_env	Sets gateway env vars
/home/bcas/item_upload_trace	A summary of all item uploads is placed in the file.
/home/bcas/item_upload_errors	A summary of all item upload errors is placed in the file.
/home/bcas/bcasitem.dat	Used to manage bcasitem queue
/home/bcas/bcasitem.idx	Used to manage bcasitem queue

## SEE ALSO

*BCASitemupload*, *qstat*, *qstatus*, *selitemcdf*, *cdf\_check*, *errtomgr*, *errtouser*, *bcasprocs\_m4*, *parsefields\_m4*, *bcashdr\_m4*

## NAME

*BCASitemupload*

## SYNOPSIS

*BCASitemupload* -t timeout -l BCASaccount <input cdf file>

## DESCRIPTION

*BCASitemupload* navigates BCAS menus to arrive at the ITEM RECORD screen. Next, it places the item, manufacturer, and part number data into the appropriate positions on the screen, then commits the changes. Errors are reported to standard output, are noted by *upload\_bcas\_item\_desc\_cron*, then mailed to the



gatecmgr. Error messages are mailed with the *errtomgr* and *itemerrtouser* Bourne Shell scripts.

## INTERNAL DESCRIPTION

The BCASitemupload gateway script takes an item description CDF file (format described later in this description) and uploads its contents into BCAS. This document is meant to be used in tandem with the comments which are in the BCASitemupload file.

After setting up the output log files, BCASitemupload calls function parsefields to read in the content of the item description CDF file (which is specified to BCASitemupload as the 5th parameter on the execute line). In order for parsefields to work, variable names identical to those of the variable names specified in the CDF file must be declared in BCASitemupload, so those variables may be correctly assigned with their respective data. Once this is accomplished, an attempt is made to connect to the WANG. If this is successful, the function bcaslogin is used to enter BCAS.. Next the BCAS screens are navigated via use of the pfkey function to reach the item description entry screen. On entry to each new screen, checks are made to insure the appropriate responses are being made by BCAS via the search for expected keywords in the term statements preceding the calls to the pfkey functions. If at any time erroneous responses are detected, the function failure\_notice is used to mail a transcript record of the screen interactions (up to the point of error) to the gatecmgr for analysis.

When the item description input screen is reached, the stock number of the item is input to BCAS (this is obtained from the data in the cdf file). BCASitemupload can detect several error situations which might arise at this interaction point; namely: item being held by another process, item not found, and illegal item number. In all cases an appropriate error message is generated to the transcript files which will be mailed to the gatecmgr.

If the item number was a legal one, the existing item description is displayed on the next screen. Actually only the first six item description lines are displayed on this first screen. Each succeeding item description screen reveals an additional fourteen lines of item description (up to a maximum of 58 lines). Unfortunately, due to the manner in which the 3rd and 4th item description screens were designed, one cannot tell the difference between them and the 2nd screen (i.e. there are no features written to the screen which are different between the 2nd, 3rd, and 4th screens). This prevents BCASitemupload from being able to access these 3rd and 4th screens, for it would never know if the 3rd or 4th screen "came up" successfully. Due to this difficulty BCASitemupload is limited to replacing only item descriptions which contain less than or equal to

eighteen lines of forty character text. Fortunately almost all item descriptions never even come close to using eighteen lines of text.

Initially, the key question BCASitemupload must answer is whether, the second item description screen is going to be needed. If BCASitemupload detects at least one blank item description line on the first screen, this will not be necessary ( since item descriptions are not allowed to have blank lines in them); i.e. one blank line implies the item description ended. In this case the next question which must be answered is whether lines must be deleted from the existing item description i.e. the new item description is shorter than the original. The general algorithm that is followed is that new lines overwrite old lines, when all the new lines have been written, if old lines still exist, they are overwritten with blank lines. This is done until all the old text has been overwritten. The same algorithm is applied to the second screen (when item descriptions using more than six lines are encountered).

When the initial item description screen has been displayed; the first modifications will be to input the manufacturer name and part number for the item. In order to insure that all characters that were in these existing fields are overwritten (usually these fields are blank), the strings which are output are extended to 30 characters (for manufacturer) and 20 characters (for part number), by concatenating spaces to the text with the function make\_space. When each item is input a check is made to insure the data was accepted by BCAS (via term statements preceding the sending of text) If not, appropriate error messages are generated for the transcript file that will be mailed to gatecmgr

After these preliminary data items are input, the main loop of item description input is entered. The function get\_next\_line is used to obtain the next line of item description text (that was read in from the cdf file). If it is non blank, the function make\_space is called to concatenate spaces onto the string to insure all previous text of the original line of item description will be overwritten when this string is written to BCAS. Each time a new line is input a check is made to insure the line was accepted by BCAS. Next, a check is made to see if it is necessary to go to the next item description screen. This is done by evaluating whether the original item description screen had any blank lines and whether the current line output was line six.. The item description output loop is terminated when all new item description lines have been input and all old lines have been erased.

After the item description input has been entered successfully, the pfkey function is used again to navigate back out of the BCAS menus. In order for the script which makes use of BCASitemupload to ascertain the status of the upload attempt, BCASitemupload will output three types of messages.

A message which has the text "upload succeeded"

A message which has the text "REQUEUE"

A message which has other text.

If the transcript file has "upload succeeded" in it; this indicates no errors were encountered in the upload and the upload was a success. If the text "REQUEUE" is present this indicates that an error was encountered that was not related to the content of the CDF upload file (e.g. BCAS went down, an unknown screen appeared during menu traversal). In this case the file is requeued by the calling script so the upload attempt can be made again. If other error text is output; this implies that re-queuing is likely to result in success and that the situation should be examined by the gatec manager before another upload attempt is made on the file.

## TESTING

Developing a text fixture for *BCAScitemupload* would require a series of tests which insure that all error conditions nominally encountered for item upload are handled correctly. Error conditions which might occur include,

attempt to mail error message failed

error deleting file

illegal cdf file name

error from parsefileds routine

connection refused

on unexpected BCAS screen

error getting current date

unable to go to BCAS MENU screen

unable to go to FILE MAINTENANCE screen

unable to go to ITEM MASTER MENU screen

stock number being held

stock number not found

illegal number

unable to go to ITEM RECORD screen

unable to input manufacturer name

unable to input part number name

unable to input item description line

did not successfully read next line of item description

unable to go to additional item screens

cannot go to third item description screen

To make sure each condition is handled correctly, each error is made to occur, then the output results can be verified.

Another test would be to input test item description files having item descriptions varying in length from one to eighteen lines, replacing an existing item description with one line of text. Next, this would then be done for existing item descriptions with descriptions of 2, 3,..., and 18 lines. If the new data is updated correctly in BCAS, this would be strong evidence that the software is operating correctly.

## BCAS Item Upload CDF File Format

An example of an item CDF file is shown below,

```
%Xbegin
%Xpurpose    item upload
%Xfilename   ITEM CDF
%Xdestination_host  gatec.dui
%Xversion    2.4
%Xdate       93 05 26
%stock_number      7320PTEST2
%suffix
%unit_of_issue
%bsp   GIR
%primary_customer
%variation
%automatic_po
%brand_name_sole_source
%commodity_assignment
%manufacturers_name      RICOH1
%manufacturers_partno    SM300034-1
%description01            THIS IS THE FIRST LINE
%description02
%description03
%description04
%description05
%description06
%description07
%description08
%description09
%description10
%description11
%description12
%description13
%description14
%description15
%description16
%description17
%description18
%description19
%description20
%description21
%description22
%description23
%description24
%description25
%description26
%description27
%description28
```

%description29  
%description30  
%description31  
%description32  
%description33  
%description34  
%description35  
%description36  
%description37  
%description38  
%description39  
%description40  
%description41  
%description42  
%description43  
%description44  
%description45  
%description46  
%description47  
%description48  
%Xend

## **BCASitemupload Variable Definitions**

### **Global Variables used as Returned Parameters from Procedures**

string space\_string - returned by procedure make\_space. Will contain the number of spaces requested to be placed in string.

string line\_to\_output - returned by procedure get\_next\_line. Will contain the next item description line from the CDF file.

### **Global Constants**

int total\_chars\_for\_man\_name - Number of characters BCAS allows for manufacturer name in the item description (30).

int total\_chars\_for\_part\_no - Number of characters BCAS allows for part number in the item description (20).

int total\_chars\_for\_item\_desc - Maximum number of characters for each item description line of text (40).

string \_me - Identifies gateway script (GW) for error message output.

### **Variables Used by Procedure Parsefields to Hold Item CDF File Contents**

string Xbegin - CDF related.

string Xpurpose - CDF related.

string Xfilename - CDF related.

string Xdestination\_host - CDF related.

string Xversion - CDF related.

string Xdate - CDF related.

string stock\_number - used to pull up item description in BCAS.

string suffix - not used (FSC suffix).

string unit\_of\_issue - not used (e.g. EA, PG, etc.).

string bsp - not used (buyer e.g. G1R).

string primary\_customer - not used.

string variation - not used.

string automatic\_po - not used.

string brand\_name\_sole\_source - not used.

string commodity\_assignment - not used.

string manufacturers\_name - Updated.

string manufacturers\_partno - Updated.

string description01 - begin item description.

string description02

string description03

string description04  
string description05  
string description06  
string description07  
string description08  
string description09  
string description10  
string description11  
string description12  
string description13  
string description14  
string description15  
string description16  
string description17  
string description18  
string description19  
string description20  
string description21  
string description22  
string description23  
string description24  
string description25  
string description26  
string description27  
string description28  
string description29  
string description30  
string description31  
string description32  
string description33  
string description34  
string description35  
string description36  
string description37  
string description38  
string description39  
string description40  
string description41  
string description42  
string description43  
string description44  
string description45  
string description46  
string description47  
string description48  
string Xend - CDF related

## **File I/O**



file recordpipe - i/o channel for file with name bcas-item-upload-record. A detailed transcript record kept in /home/bcas directory.

file logpipe - i/o channel for file with name of form <month>-<day>-<year>-<hr>-<min>-<sec>-item-log (e.g. 05-18-93-17-22-10-item-log) hold summary information for BCAS interaction. Kept in /tmp directory.

file input - i/o channel for input CDF file.

string fname - file name of cdf file currently processing.

string log\_fname - file name for item log file.

### **Variables Used to Monitor Initial Item Description Screen Display**

int end\_of\_first\_screen - a 'ITEM RECORD' string and 'PRINT SCREEN' string (output as BCAS paints the current screen) have been detected.

int blank\_item\_lines - number of blank lines of item description detected.

int item\_record\_string\_found - a 'ITEM RECORD' string (output as BCAS paints the current screen) has been detected.

### **Variables Used to Correctly Output Manufacturers Name and Part Number**

int num\_man\_chars - current number of characters in manufacturers name just read from cdf.

string num\_blank\_spaces - number of blank spaces needed to finish out the line after the manufacturers name has been output.

num\_part\_no\_chars - current number of characters in part number just read from cdf.

### **Variables Used in Item Description Fill In Loop**

string continue\_item\_input - perpetuates main loop which will output item description text.

string current\_item\_line - current line number for which text will be entered. NOTE this variable is reset to 1 every time an additional item screen is pulled up.

string current\_item\_line\_from\_cdf - current line have read from cdf (max 48).

int on\_first\_item\_screen - indicates if we are on the first item description screen. If so, then max item lines to enter will be 6.

int max\_lines\_on\_1st\_item\_screen - 6.

int max\_lines\_on\_other\_item\_screen - 14.

string num\_line\_no\_chars - Number of characters in the item desc line being output.

string row\_on - line number after next item line has been input.

string col\_on - column number cursor will be on after next item line input.

29 on first screen

23 on subsequent screens

string response\_str - string to look for after an item description line has been input.

string item\_screen\_count - max 3 additional item screens

## ENVIRONMENT

\$ECLIB must be set. See FILES

## FILES

\$ECLIB/gatewaylog	File where gateway puts log messages
\$ECLIB/coredir	Directory where gateway puts core dumps
/etc/.authlist	File containing encrypted password for the BCAS account specified with in the -l option.
/home/bcas/item-upload-record	Diagnostic gateway output for all item upload transactions is placed in this file
/tmp/<month>-<day>-<year>-<hr>-<min>-<sec>-item-log	Holds summary information for BCAS interaction (e.g. 05-18-93-17-22-10-item-log)

## SEE ALSO

*BCASitemupload, qstat, qstatus, selitemcdf, cdf\_check, errtomgr, errtouser, bcasprocs\_m4, parsefields\_m4, bcashdr\_m4*

#### NAME

*selitemcdf*

#### SYNOPSIS

cat <item\_cdf\_file> |selitemcdf

#### DESCRIPTION

Outputs specified items CDF parameters to stdout.

#### INTERNAL DESCRIPTION

names array sets up legal CDF parameters to be expected in CDF file. Each line in the CDF file is examined. If data is detected for a parameter, it is output to stdout.

#### SEE ALSO

*upload\_bcas\_item\_desc\_cron, upload\_cancel\_award\_cron, cdf\_check*

#### NAME

*itemerrtouser*

#### SYNOPSIS

<message> | itemerrtouser <users>

#### DESCRIPTION

*itemerrtouser* reads stdin and directs all data found there into a mail message and forwards that message to the indicated users.

#### SEE ALSO

*cancelerrtouser, errtomgr, upload\_bcas\_item\_desc\_cron, upload\_cancel\_award\_cron*

---

## 1.6 BCAS Cancel Award Upload

---

Source files:

upload\_bcas\_cancel\_award\_cron, BCAScancelaward, errotmgr, cancelerrtouser, selcancelcdf

Utilities used:

qstatus, qpop, qadditem, cdf\_check

NAME

*upload\_bcas\_cancel\_award\_cron*

SYNOPSIS

upload\_bcas\_cancel\_award\_cron

DESCRIPTION

Any time a purchase order is canceled by a buyer using the GATEC application, the cancellation information is uploaded back to BCAS via the Bourne Shell script *upload\_bcas\_cancel\_award\_cron*. When award cancellation is committed by a buyer, the GATEC application writes a CDF file containing the cancellation details (obtained from the cancellation screen ) and places that file on the *bcascancel* queue. An 850 and 836 CDF are also generated and their names are placed on the *bcascancel* queue at this time. *upload\_bcas\_cancel\_award\_cron* takes the cancel CDF filenames off this queue (using the *qpop* utility), insures the data in them is correct (using *selcancelcdf* and *cdf\_check*), then passes the filename to the gateway script *BCAScancelaward*. *BCAScancelaward* updates BCAS with the cancellation information.

INTERNAL DESCRIPTION

upload\_bcas\_cancel\_award\_cron makes use of gateway script *BCAScancelaward* to upload the contents of cancel CDF files (whose filenames reside on the *bcascancel* queue) back to BCAS.

After setting up file aliases and insuring that another version of itself is not running, *upload\_bcas\_cancel\_award\_cron* sets up its own environment, then makes use of *qstatus* to insure the *bcasitem* item queue is operational. At this point the main loop is entered

where the names of CDF files to be processed are repeatedly popped off the *bcascancel* queue via the *qpop* utility. After a check the integrity of the information popped off the queue (i.e. valid key and data), the existence of the CDF file is confirmed. Next, the Perl scripts *selcancelcdf* and *cdf\_check* are used to check to syntax of specified fields in the file. If the syntax checks are passed, *BCASccancelaward* is called with the name of the CDF file whose data is to be uploaded to BCAS.

After *BCASccancelaward* execution a *grep* is made of its output. If the keywords "award cancel succeeded" are detected, the corresponding 850CDF is sent to the outbound queue for generation of an X12 850 cancellation, then the name of the next file to process is popped off the queue. If the success string is not found, then a check is made to determine if more than three consecutive errors have occurred, if so the cron script is stopped. If the error count is less than three, a check is made to determine whether *BCASccancelaward* has recommended requeuing the file. If so (and it has not been requeued once), the file is requeued, otherwise a check is made to see whether the BCAS failure was because of a disconnect. If this was the case the file is requeued and the next file is processed--otherwise the troubled file will not be requeued and the next file will be examined by popping the *bcascancel* queue.

Processing (for non error situations) continues until the *bcascancel* queue is empty. Bourne Shell scripts *cancelerrortouser*, and *errtomgr* are used to report errors to the gatec manager.

## TESTING

Developing a test fixture for this software is very straightforward. The following error conditions are detected:

lockfile exists (version of *upload\_bcas\_cancel\_award\_cron* already running)

cannot touch lockfile

cannot touch summary file

*qstatus* unknown exit error

queue not up

queue not empty

cdf filename null

cdf file does not exist

grammar error in CDF file

unable to send 850 CDF out to outbound queue

more than three consecutive errors detected in upload

REQUEUE message from *BCAScancelaward* detected

DISCONNECT message from *BCAScancelaward* detected

failed requeue

To make sure each condition is handled correctly, each error is made to occur, then the output results are analyzed.

## FILES

/home/bcas/\$LOCKFILE	Lock file
\$HOME/set_ecedi_env	Sets gateway env vars
/home/bcas/cancel_award_trace	A summary of all cancel uploads is placed in this file.
/home/bcas/cancel_award_errors	A summary of all cancel errors is placed in the file
cancel_award_record	Diagnostic gateway output for all cancel transactions is placed in this file.
/home/bcas/que/bcascancel.dat	Used to manage bcascancel queue
/home/bcas/que/bcascancel.idx	Used to manage bcascancel queue

## SEE ALSO

*BCAScancelaward*, *qstat*, *qstatus*, *selcancelcdf*, *cdf\_check*, *errtomgr*, *errtouser*, *bcasprocs\_m4*, *parsefields\_m4*, *bcashdr\_m4*

NAME

*BCAScancelaward*

SYNOPSIS

*BCAScancelaward* -t timeout -l BCASaccount <input cdf file>

DESCRIPTION

*BCAScancelaward* navigates BCAS menus to arrive at the MODIFICATION ACTION screen. Next, it places the cancel details onto this screen as well as the MODIFICATION RESULT, and NARRATIVE screens. If the cancellation information was successfully uploaded to BCAS, the associated 850 CDF file is placed on a printer queue known as "outbound", (via `lpr -Poutbound <filename>`). where it will be subsequently processed by the *outbound* script on the Transport Machine. The 836 CDF is not used in this application.

INTERNAL DESCRIPTION

The *BCAScancelaward* gateway script takes an item description CDF file (format described later in this description) and uploads its contents into BCAS. This document is meant to be used in tandem with the comments which are in the *BCAScancelaward* file.

After setting up the output log files, *BCASitemupload* calls function `parsefields` to read in the content of the cancel description CDF file (which is specified to *BCAScancelaward* as the 5th parameter on the execute line). In order for `parsefields` to work, variable names identical to those of the variable names specified in the CDF file must be declared in *BCAScancelaward*, so those variables may be correctly assigned with their respective data. Once this is accomplished, an attempt is made to connect to the WANG. If this is successful, the function `bcaslogin` is used to enter BCAS.. Next the BCAS screens are navigated via use of the `pfkey` function to reach the modification action screen. On entry to each new screen, checks are made to insure the appropriate responses are being made by BCAS via the search for expected keywords in the term statements preceding the calls to the `pfkey` functions. If at any time erroneous responses are detected, the function `walk_out` is used to mail a transcript record of the screen interactions (up to the point of error) to the `gatecmgr` for analysis.

When the modification action screen is reached the following data is taken from the input CDF and input to that BCAS screen:

award piin  
supplemental piin  
activity number  
buyer code  
modification reason  
effective date  
Final/Temp/Draft  
contractor sign  
number of copies  
suspense date  
cancel order entirely  
cancel prs to customer  
reopen prs  
order statement one  
order statement two  
order statement three  
order statement four  
order statement five  
order statement six  
order statement seven  
order statement eight  
order statement nine  
order statement ten  
special prep  
reopen prs



For these parameters a check is made to see whether the data is actually specified in the CDF; if so, the data is sent to BCAS, otherwise a horizontal tab is entered to get to the next data entry point. Regardless, a check is made (via a term statement) to make sure the screen cursor is in the next expected position after the input of the data or the tab.

Once the data on the screen is entirely entered, it is sent to BCAS for processing. Many error can be detected by BCAS at this point; specifically,

award piin not loaded

item delivered cannot cancel

field cannot be blank

improper buyer id

1st character not alpha

2nd character not 1-9

3rd character not alpha

invalid modification reason

invalid date input

invalid month input

must use D, F, or T (for selection of Draft, Final, or Temp)

contractor sign should be N when T used

field must be Y or N

since contractor sign was no this field should be blank

since contractor sign was yes, this field should be ALL or 1-6

since draft suspense dat should be specified

suspense date must be input

Y, N, or spaces need to be used for cancel/reopen prs fields

invalid order statement used

cancel prs customer and with reopen prs customer cannot be the same

If the data are accepted and the requisition is not being reopened, the modify order level data screen comes up next. Nothing is actually done on this screen except to move to the modification result screen where the narrative information is input (sections c, d, and the actual narrative).

Input of the narrative first requires that the length of the narrative to be accessed, this is done by using `get_next_narr_line` to see how many lines the narrative spans. This can be somewhat tricky since the narrative can have blank lines in it. The algorithm keeps track of where the last text line was, before the final blank lines are detected preceding the end of narrative. Once this is calculated the lines are input one at a time, again using `get_next_narr_line` to access the narrative text. After the narrative is input, the cancel is committed.

## TESTING

Developing a text fixture for *BCAScancelaward*, would require a series of tests which insure that all error conditions nominally encountered for cancel award are handled correctly. In addition to the error conditions mentioned above, other error conditions which might occur include,

attempt to mail error message failed

error deleting file

illegal cdf file name

error from parsefields routine

connection refused

on unexpected BCAS screen

error getting correct date

unable to go to BCAS MENU screen

unable to go to CONTRACT ADMINISTRATION screen

unable to go to MODIFICATION MENU screen

unable to go to MODIFICATION ACTION screen

unsuccessful input of award piin  
unsuccessful input of supplemental piin  
unsuccessful input of activity number  
unsuccessful input of buyer code  
unsuccessful input of modification reason  
unsuccessful input of effective date  
unsuccessful input of Final Temp or Draft  
unsuccessful input of contractor sign  
unsuccessful input of number of copies  
unsuccessful input of suspense date  
unsuccessful input of cancel entirely  
unsuccessful input of cancel prs to customer  
unsuccessful input of reopen prs  
unsuccessful input of order statement one  
unsuccessful input of order statement two  
unsuccessful input of order statement three  
unsuccessful input of order statement four  
unsuccessful input of order statement five  
unsuccessful input of order statement six  
unsuccessful input of order statement seven  
unsuccessful input of order statement eight  
unsuccessful input of order statement nine  
unsuccessful input of order statement ten  
unsuccessful input of special preparation  
did not reach begin narrative screen

did not reach description of amendment screen

could not reach c in supplemental narrative screen

could not reach d in supplemental narrative screen

could not reach begin narrative section

did not make it to next narrative line

To make sure each condition is handled correctly, each error is made to occur, then the output results can be verified.

## BCAS Cancel Award CDF File Format

An example of a cancel CDF is shown below,

```
%Xbegin
%Xpurpose      cancel award upload
%Xfilename     CANCEL AWARD CDF
%Xdestination_host  gatec.dui
%Xversion      2.4
%Xdate 93 12 15
%award_piin    94M6132
%supp_piin
%activity_no
%cac  G1D
%mod_reason    W
%effective_date 93DEC15
%f_t_d F
%contractor_signs  N
%number_of_copies
%suspense_date
%cancel_entirely  Y
%cancel_prs_cust  N
%with_reopen_prs  N
%order_statement01
%order_statement02
%order_statement03
%order_statement04
%order_statement05
%order_statement06
%order_statement07
%order_statement08
%order_statement09
%order_statement10
%special_prep  N
%new_solit_number
%contract_number
%narrative01  1. SUBJECT ORDER IS HEREBY CANCELLED IN ITS
%narrative02  ENTIRETY AS IT WAS AWARDED IN ERROR.
%narrative03
%narrative04  2. AS A RESULT OF THE ABOVE MODIFICATION, THE
%narrative05  TOTAL AMOUNT IS DECREASED FROM $43.47 TO $0.00
%narrative06  FOR A TOTAL DECREASE OF $43.47.
%narrative07
%narrative08
%narrative09
%narrative10
%narrative11
%narrative12
%narrative13
%narrative14
```

%narrative15  
%narrative16  
%narrative17  
%narrative18  
%narrative19  
%narrative20  
%narrative21  
%narrative22  
%narrative23  
%narrative24  
%narrative25  
%narrative26  
%narrative27  
%narrative28  
%narrative29  
%narrative30  
%narrative31  
%narrative32  
%no\_reopen\_cr M  
%w\_reasons  
%Xend

## **BCASitemupload Variable Definitions**

### **Global Variables used as Returned Parameters from Procedures**

current\_narrative\_line - returned by procedure get\_next\_line. Will contain the next narrative line from the CDF file.

### **Global Constants**

string \_me - Inites gateway script (GW) for error message output.

### **Variables Used by Procedure Parsefields to Hold Cancel CDF File Contents**

string Xbegin - CDF related.  
string Xpurpose - CDF related.  
string Xfilename - CDF related.  
string Xdestination\_host - CDF related.  
string Xversion - CDF related.  
string Xdate - CDF related.  
string award\_piin - The award piin associated with the RFQ  
string supp\_piin - Normally not used.  
string activity\_no - Normally not used.  
string cac - The buyer code.  
string mod\_reason - ( a - z --- see BCAS manual)  
string effective\_date - Current date  
string f\_t\_d - Whether this cancellation is in final, temporary, or draft form.  
string contractor\_signs - Whether the contractor whose award is being canceled needs to sign the form.  
string number\_of\_copies - Number of printed copies of the cancellation to generate.  
string suspense\_date - Need to specify for temporary or draft.  
string cancel\_entirely - Whether the entire order is to be canceled t.  
string cancel\_prs\_cust -  
string with\_reopen\_prs - Is RFQ to be re-opened?  
string order\_statement01 - Normally not used.  
string order\_statement02 - Normally not used.  
string order\_statement03 - Normally not used.  
string order\_statement04 - Normally not used.  
string order\_statement05 - Normally not used.  
string order\_statement06 - Normally not used.  
string order\_statement07 - Normally not used.  
string order\_statement08 - Normally not used.  
string order\_statement09 - Normally not used.  
string order\_statement10 - Normally not used.  
string special\_prep - Any special preparation required.  
string new\_solit\_number - If old RFQ is to be re-opened it needs an new RFQ number.  
string contract\_number - Need to specify GSA contract number of award canceling was a GSA award.  
string narrative01 - Narrative describing details of cancellation.  
string narrative02 - Narrative describing details of cancellation.

string narrative03 - Narrative describing details of cancellation.  
string narrative04 - Narrative describing details of cancellation.  
string narrative05 - Narrative describing details of cancellation.  
string narrative06 - Narrative describing details of cancellation.  
string narrative07 - Narrative describing details of cancellation.  
string narrative08 - Narrative describing details of cancellation.  
string narrative09 - Narrative describing details of cancellation.  
string narrative10 - Narrative describing details of cancellation.  
string narrative11 - Narrative describing details of cancellation.  
string narrative12 - Narrative describing details of cancellation.  
string narrative13 - Narrative describing details of cancellation.  
string narrative14 - Narrative describing details of cancellation.  
string narrative15 - Narrative describing details of cancellation.  
string narrative16 - Narrative describing details of cancellation.  
string narrative17 - Narrative describing details of cancellation.  
string narrative18 - Narrative describing details of cancellation.  
string narrative19 - Narrative describing details of cancellation.  
string narrative20 - Narrative describing details of cancellation.  
string narrative21 - Narrative describing details of cancellation.  
string narrative22 - Narrative describing details of cancellation.  
string narrative23 - Narrative describing details of cancellation.  
string narrative24 - Narrative describing details of cancellation.  
string narrative25 - Narrative describing details of cancellation.  
string narrative26 - Narrative describing details of cancellation.  
string narrative27 - Narrative describing details of cancellation.  
string narrative28 - Narrative describing details of cancellation.  
string narrative29 - Narrative describing details of cancellation.  
string narrative30 - Narrative describing details of cancellation.  
string narrative31 - Narrative describing details of cancellation.  
string narrative32 - Narrative describing details of cancellation.  
string no\_reopen\_cr - If RFQ not to be re-opened must specify a reason

J - Cancel per customer request

K - Unilaterally terminated

M - Vendor refuses to effect terms and conditions of order.

string w\_reasons  
string Xend

## **File I/O**

string lname - Holds log on name parameters specified on BCAScancelaward run line

string bcasSystem - Holds name of BCAS system

string log\_fname - File name for cancel log file

string fname - File name of input CDF file.



string timeout - Holds time out parameter specified on BCAScancelaward run line.

file recordpipe - i/o channel for file with name bcas\_cancel\_award\_record. A detailed transcript record kept in /home/bcas directory.

file logpipe - i/o channel for file with name of form <month>-<day>-<year>-<hr>-<min>-<sec>-cancel-log (e.g. 05-18-93-17-22-10-item-log) hold summary information for BCAS interaction. Kept in /tmp directory.

file input - i/o channel for input CDF file.

### **Variables Used to Ascertain Length of Narrative Line**

int firstBlank - (1-currently processing group of 1 or more blank lines).

int lineBlankStarted - line that the current set of blanks started on.

string lastLine - the previous line read from the narrative.

int noBlanks - current number of consecutive blank lines encountered.

string current\_narrative\_line - increments index passed to get\_next\_narr\_line to get next line.

### **Variables Used In Output of Narrative Line**

string response\_str - Holds expected cursor position after input of current narrative line to BCAS.

## **ENVIRONMENT**

\$ECLIB must be set. See FILES

## **FILES**

\$ECLIB/gatewaylog	File where gateway puts log messages
\$ECLIB/coredir	Directory where gateway puts core dumps
/etc/.authlist	File containing encrypted password for the
	BCAS account specified with in the -l option.
/home/bcas/cancel_upload_record	Diagnostic gateway output for all item upload
	transactions is placed in this file
/tmp/<month>-<day>-<year>-<hr>-<min>-<sec>-cancel-log	
Holds summary information for BCAS interaction (e.g. 05-18-93-17-22-10-cancel-log)	

## **SEE ALSO**

*BCAScancelaward, qstat, qstatus, selcancelcdf, cdf\_check, errtomgr, bcasprocs\_m4, parsefields\_m4, bcashdr\_m4*

## NAME

*selcancelcdf*

## SYNOPSIS

cat <cancel\_cdf\_file> |selcancelcdf

## DESCRIPTION

Outputs specified cancel CDF parameters to stdout.

## INTERNAL DESCRIPTION

names array sets up legal CDF parameters to be expected in CDF file. Each line in the CDF file is examined. If data is detected for a parameter, it is output to stdout.

## SEE ALSO

*upload\_bcas\_item\_desc\_cron*, *upload\_cancel\_award\_cron*,  
*cdf\_check*

## NAME

*cancelerrtouser*

## SYNOPSIS

<message> | cancelerrtouser <users>

## DESCRIPTION

*cancelerrtouser* reads stdin and directs all data found there into a mail message and forwards that message to the indicated users.

## SEE ALSO

*itemerrtouser*, *errtomgr*, *upload\_bcas\_item\_desc\_cron*,  
*upload\_cancel\_award\_cron*

NAME

*errtomgr*

SYNOPSIS

<message> | *errtomgr* [<users>]

DESCRIPTION

*errtomgr* reads stdin and directs all data found there into a mail message and forwards that message to *thegatecmgr* as well as to other users (if they are specified).

OPTIONS

<users>            Other users mail message should go to.

SEE ALSO

*cancelerrtouser*,    *itemerrtouser*,    *errtomgr*,  
*upload\_bcas\_item\_desc\_cron*, *upload\_cancel\_award\_cron*

NAME

*errtouser* - mail stdin to *gatecmgr*, and optionally to some user.

SYNOPSIS

*errtouser* [ subject ]

DESCRIPTION

*errtouser* reads from stdin, and mails all input to the user *gatecmgr*, setting the Subject: line to subject if it is specified, else setting the SUBJECT: line to none.

if a line of the form

%buyer\_code user

exists in stdin, mail is also sent to user.

## SEE ALSO

*putuploads\_cron(1)*

## NAME

*errtomgr* - mail stdin to gatecmgr

## SYNOPSIS

*errtomgr* [ subject ]

## DESCRIPTION

*errtomgr* reads from stdin, and mails all input to the user *gatecmgr*, setting the Subject: line to subject if it is specified.

## NAME

*cdf\_check* - syntax check a cdf file

## SYNOPSIS

*cdf\_check* [ filename . . . ]

## DESCRIPTION

*cdf\_check* checks that files contain valid cdf regular expressions. If no files are specified, *cdf\_check* assumes standard input. *cdf\_check* exits 1 if there are any syntactically incorrect regular expressions, else it exits 0.

## FILES

~gatec2/etc/cdf\_regexp  
File of cdf regular expressions.

## SEE ALSO

*cdf\_regexp(5)*, *perl(1)*

## NAME

*cdf\_regex* - file of valid regular expressions for cdf files.

## DESCRIPTION

cdf files contain name-value pairs. For each name, *cdf\_regex* contains a regular expression, in Perl format. This is used by the program *cdf\_check* to verify that a cdf file contains correct values. Currently this file is used only for validating cdf files that contain information to be used to make an award, upload an item description, or cancel an award on the Wang BCAS system.

## SEE ALSO

*cdf\_check(1)*, *cdf\_regex\_big(5)*, *perl(1)*

## NAME

*regcomp*, *regexexec*, *regsub*, *regerror* - regular expression handler

## SYNOPSIS

```
#include <v8regex.h>
```

```
regex *regcomp(exp)
char *exp;
```

```
int regexexec(prog, string)
regex *prog;
char *string;
```

```
regsub(prog, source, dest)
regex *prog;
char *source;
char *dest;
```

```
regerror(msg)
char *msg;
```

## DESCRIPTION

These functions implement *egrep*(1)-style regular expressions and supporting facilities.

*Regcomp* compiles a regular expression into a structure of type *regex*, and returns a pointer to it. The space has been allocated using *malloc*(3) and may be released by *free*.

*Regexexec* matches a NUL-terminated string against the compiled regular expression in *prog*. It returns 1 for success and 0 for failure, and adjusts the contents of *prog*'s *startp* and *endp* (see below) accordingly.

The members of a *regex* structure include at least the following (not necessarily in order):

```
char *startp[NSUBEXP]; char *endp[NSUBEXP];
```

where *NSUBEXP* is defined (as 10) in the header file. Once a successful *regexexec* has been done using the *regex*, each *startp*-*endp* pair describes one substring within the string, with the *startp* pointing to the first character of the sub-string and the *endp* pointing to the first character following the substring. The 0th substring is the substring of string that matched the whole regular

expression. The others are those substrings that matched parenthesized expressions within the regular expression, with parenthesized expressions numbered in left-to-right order of their opening parentheses.

*Regsub* copies source to dest, making substitutions according to the most recent regexexec performed using prog. Each instance of `&' in source is replaced by the substring indicated by startp[0] and endp[0]. Each instance of `n', where n is a digit, is replaced by the substring indicated by startp[n] and endp[n].

To get a literal `&' or `n' into dest, prefix it with `\''; to get a literal `\' preceding `&' or `n', prefix it with another `\'.

*Regerror* is called whenever an error is detected in regcomp, regexexec, or regsub. The default regerror writes the string msg, with a suitable indicator of origin, on the standard error output and invokes exit(2). Regerror can be replaced by the user if other actions are desirable.

## REGULAR EXPRESSION SYNTAX

A regular expression is zero or more branches, separated by `|'. It matches anything that matches one of the branches.

A branch is zero or more pieces, concatenated. It matches a match for the first, followed by a match for the second, etc.

A piece is an atom possibly followed by `\*', `+', or `?'. An atom followed by `\*' matches a sequence of 0 or more matches of the atom. An atom followed by `+' matches a sequence of 1 or more matches of the atom. An atom followed by `?' matches a match of the atom, or the null string.

An atom is a regular expression in parentheses (matching a match for the regular expression), a range (see below), `.' (matching any single character), `^' (matching the null string at the beginning of the input string), `\$' (matching the null string at the end of the input string), a `\' followed by a single character (matching that character), or a single character with no other significance (matching that character).

A range is a sequence of characters enclosed in `[]'. It normally matches any single character from the sequence. If the sequence begins with `^', it matches any single character not from the rest of the sequence. If two characters in the sequence are separated by `-', this is shorthand for the full list of ASCII characters between them (e.g. `[0-9]' matches any decimal digit). To include a literal `]' in the sequence, make it the first character (following a possible `^').

To include a literal '-', make it the first or last character.

## AMBIGUITY

If a regular expression could match two different parts of the input string, it will match the one which begins earliest. If both begin in the same place but match different lengths, or match the same length in different ways, life gets messier, as follows.

In general, the possibilities in a list of branches are considered in left-to-right order, the possibilities for '\*', '+', and '?' are considered longest-first, nested constructs are considered from the outermost in, and concatenated constructs are considered leftmost-first. The match that will be chosen is the one that uses the earliest possibility in the first choice that has to be made. If there is more than one choice, the next will be made in the same manner (earliest possibility) subject to the decision on the first choice. And so forth.

For example, '(ab|a)b\*c' could match 'abc' in one of two ways. The first choice is between 'ab' and 'a'; since 'ab' is earlier, and does lead to a successful overall match, it is chosen. Since the 'b' is already spoken for, the 'b\*' must match its last possibility-the empty string-since it must respect the earlier choice.

In the particular case where no '|'s are present and there is only one '\*', '+', or '?', the net effect is that the longest possible match will be chosen. So 'ab\*', presented with 'xabbbby', will match 'abbbb'. Note that if 'ab\*' is tried against 'xabyabbbz', it will match 'ab' just after 'x', due to the begins-earliest rule. (In effect, the decision on where to start the match is the first choice to be made, hence subsequent choices must respect it even if this leads them to less-preferred alternatives.)

## SEE ALSO

*egrep(1)*, *expr(1)*

## DIAGNOSTICS

*Regcomp* returns NULL for a failure (regerror permitting), where failures are syntax errors, exceeding implementation limits, or applying '+' or '\*' to a possibly-null operand.

## HISTORY

Both code and manual page were written at U of T. They are intended to be compatible with the Bell V8 regexp(3), but are not derived from Bell code.



## BUGS

Empty branches and empty regular expressions are not portable to V8. The restriction against applying `\*' or `+' to a possibly null operand is an artifact of the simplistic implementation.

Does not support egrep's newline-separated branches; neither does the V8 regexp(3), though.

Due to emphasis on and simplicity, it's not strikingly fast. It does give special attention to handling simple cases quickly.

---

## 1.8 Gateway Support Software

---

### NAME

*bcaslogin, bcaslogout, checkresp, finis, pfkey, substr, fixfield, cmdargs, expresslogout* - General WANG BCAS Gateway procedures

### SYNOPSIS

call bcaslogin(account, password, btimeout, me)

call bcaslogout

call checkresp(resp)

call finis

call pfkey(choice)

call substr(s, start, end)

call fixfield(field, len)

call cmdargs(switchstr)

call expresslogout

### DESCRIPTION

*bcaslogin* takes four arguments: the Wang userid account, the password of that userid, a btimeout in seconds that specifies how long the program is to wait for prompts, and some string me that will appear in any error messages the program generates.

*bcaslogin* is called after a successful Gateway connect has been done, and performs a login (via telnet) to the Wang system. *bcaslogin* will leave you at either the SYSTEMS ADMINISTRATORS menu, the Wang VS Command Processor menu, or the VS OFFICE menu, depending on which menu the Wang System Administrator has set up for user account.

On success, the global variable *ok* is set to TRUE, and the global variable *Current\_screen* is set to one of *SAscreen*, *CPscreen*, or *VSscreen*.

*bcaslogout* logs out from the Wang from any of the screens *SAscreen*, *CPscreen*, *VSscreen*, *RUNscreen*, *DEBUGscreen*, or *GETPARMscreen*.

*checkresp* takes one argument, the string *resp*, and echoes it to stdout.

*inis* performs an exit from the Gateway program.

*pfkey* takes one argument *choice*, which is either a number >from 1 to 32, or the string "HELP". It sets the global variable *\_string* to the equivalent WANG function key.

*substr* extracts a substring from *s* starting at character *start* and ending at character *end*. The substring is placed in the global variable *\_string*.

*fixfield* takes the string argument *field* and truncates it to less than or equal to *len* characters. If the resulting field is less than *len* characters in length, then a horizontal tab is added to the end. The resulting fields is placed in the global variable *\_string*.

*cmdargs* processes command line arguments and sets the global variables *ok*, *argc*, *arg*, *BCASaccount*, *BCASpassword*, *BCASsystem*, *\_timeout*, *stock\_file*. In addition, if *switchstr* is *setCDFfilelist*, it sets *CDFfilelist*, or if *switchstr* is *set-program*, it sets *program*.

*expresslogout* calls *bcaslogout*, then calls *finis*.

SEE ALSO

*BCASrunproc(1)*, *BCASupload(1)*, *BCASitemupload(1)*,  
*BCAScancelaward(1)*

The bcasupload, bcasitem, and bcascancel queues are implemented with public domain software which is described below

#### NAME

*q* - A simple, multi-user queuing system based on the simple, multi-user database library libdb.a.

#### SYNOPSIS

```
#include <q.h>

int queadditem(char *quename, char *key, char *data)
% qadditem quename key data

int quelist(char *quename);
% qlist quename

char *quepop(char *quename, char *key)
% qpop quename

char *quekeypop(char *quename, char *key)
% qkeypop quename key

int questatus(char *quename);
% qstatus quename

int quesetstatus(char *quename, int state)
% qsetstatus quename state

% qallstatus
```

#### DESCRIPTION

It's not really a queue, it's more like a collection of objects.

*queadditem()* adds an item to the queue. It returns 0 on success, -1 if key already exists, or 1 if no such queue or any other error.

*qadditem* is the shell interface.

*quelist()* lists (to stdout) all the keys and items in the queue. It returns 0 on success, 1 if no such queue

*qlist* is the shell interface.

*quepop()* returns the next item on the queue, or NULL if no more items or no such queue. Removes the item from the queue. If key is a non-null pointer, *quepop()* stores the key of the popped item in key. There is no order to the records returned by *quepop()*.

*qpop* is the shell interface. As *quepop()*, but displays key and data to stdout. Returns 0, or 1 if no more items or no such queue.

*quekeypop()* fetches and deletes record from queue *quename* by specifying its key. The return value is a pointer to the data that was stored with the key, or NULL if record not found or no such queue.

*qkeypop* is the shell interface. As *quekeypop()*, but displays data record to stdout. Returns 0, or 1 if record not found or no such queue.

*questatus()* returns UP or DOWN (defined in *q.h*), or 1 if no such queue.

*qstatus* is the shell interface. Prints *quename*, followed by UP or DOWN, to stdout, and exits with a return status of 0, or 1 if *quename* does not exist.

*questatus()* sets the state of queue *quename*. state is either UP or DOWN. Returns 0 on success, 1 on failure. Creates queue *quename* if it doesn't exist.

*qsetstatus* is the shell interface.

*qallstatus* prints, for each queue that has been created, the *quename*, followed by UP or DOWN

SEE ALSO

*db(3)*, *tisp(3)*

CAVEATS

Both key and data must be at least one byte long, and less than 1024 bytes. See *db(3)*.

NOTES

1. Written in C++ -- it might work under ANSI C (acc on the sun)
2. You'll need to set -I/ec-ed1/include for compiles.
3. You'll need to compile with the -Bstatic flag if using CC and planning to run your program on one of the Wright- Patterson Suns (they don't have the C++ shared libraries).
4. You'll need to set -L/ec-ed1/lib to search for the libraries, and you'll need to load with the libraries -ltispc -ltispcdb -ltisp.
5. For users of the queue "bcasupload", invoke as  
queadditem("bcasupload", cdf\_file\_for\_upload,  
cdf\_file\_for\_translation\_to\_850);

The queue software actually makes use of the following database library utilities to implement the queue operations described above:

## NAME

*db\_open, db\_close, db\_store, db\_fetch, db\_delete, db\_rewind, db\_nextrec* - simple database operations used by the *q* routines

## SYNOPSIS

```
#include <db.h>
```

```
DB *db_open(const char *pathname, int oflag, int mode)
```

```
void *db_close(DB *db)
```

```
int db_store(DB *db, const char *key, const char *data, int  
flag)
```

```
char *db_fetch(DB *db, const char *key);
```

```
int db_delete(DB *db, const char *key);
```

```
void db_rewind(DB *db);
```

```
char *db_nextrec(DB *db, char *key);
```

## DESCRIPTION

These functions implement a simple, multi-user database library, from W. Richard Stevens' "Advanced Programming in the UNIX

Environment".

*db\_open()* returns a pointer to a DB structure. If the DB structure cannot be created, a NULL pointer is returned. *db\_open()* opens or create a database. The arguments are the same as *open(2)*. *db\_open()* either creates a new database, or opens an existing database. A database consists of *pathname.idx*, the index file, and *pathname.dat*, the data file. *oflag* is used as the second argument to *open(2)*. *mode* is used as the third argument to *open(2)*.

*db\_close()* closes the database.

*db\_store()* stores a record in the database. Returns 0 if OK, 1 if record exists and flag *DB\_INSERT* is specified, -1 if record doesn't exist and flag *DB\_REPLACE* is specified.

*db\_fetch()* fetchs a record by specifying its key. The return value is a pointer to the data that was stored with the key, or NULL if the record was not found.

*db\_delete()* deletes a record by specifying its key. Returns 0 if OK, -1 if record is not found.

*db\_rewind()* resets to first record.

*db\_nextrec()* To go through the entire database, reading each record in turn, first call *db\_rewind*, then call *db\_nextrec* to read each sequential record. *db\_nextrec* returns pointer to data if OK, NULL on end of file. If key is a non-null pointer, *db\_nextrec* stores the key starting at that location. There is no order to the records returned by *db\_nextrec*.

SEE ALSO

*tisp* (3)

NAME

*tisp*

DESCRIPTION

Low level error handling library

SEE ALSO

---

1.10      Compilation and Installation of Interface to Legacy System Software

---

The makefile for the CVSROOT/src/wang directory is named Makefile. Similarly named makefiles are found in \$CVSROOT/tisp, \$CVSROOT/db, \$CVSROOT/que, and CVSROOT/narqdb/src/bin/readopr2. The following steps must be taken to generate the Interface to Legacy System binaries and scripts (note assumes LOCALLIB, LOCALINC, and CVSROOT have been defined);

1. Start at \$CVSROOT/tisp. make install creates libtisp.a. libtisp.a is installed into LOCALLIB tisp.h is installed into LOCALINC. This library contains generic low level error handling functions. There are no regression tests. \$CVSROOT/tisp is not GATEC-specific. That is, it could be re-used.

2. \$CVSROOT/db depends on tisp. make install creates libtispdb.a libtispdb.a is installed into LOCALLIB db.h is installed into LOCALINC db.3 is installed into LOCALMAN There is a little test program. To run it, "make t4; ./t4" There are no formal regression tests. \$CVSROOT/db is not GATEC-specific.

3. \$CVSROOT/que depends on db. make install creates libtispq.a libtispq.a is installed into LOCALLIB q.h is installed into LOCALINC q.3 is installed into LOCALMAN make install also creates 7 binaries. These binaries (and qlocal.h) should be pulled out of this make tree and placed elsewhere. The 7 binaries, qsetstatus, qadditem, qlist, qstatus, qallstatus, qpop, and qkeypop are installed into LOCALBIN. There are no formal regression tests. libtispq.a is not GATEC-specific. The binaries (and qlocal.h) are GATEC-specific.

4. \$CVSROOT/narqdb/src/bin/readopr2 depends on libnarq, libnora, and \$NARQDB/include. libnarq and libnora are not discussed here. readopr2 also depends on ../../../../dui/src/cdfdb/tempfile.o, which is currently known as \$CVSROOT/dui/src/cdfdb/tempfile.o. make install installs one binary, readopr2, into LOCALBIN. There is a minimalist regression test. Run it with "make test". If it returns 0 errors, you've probably set up the database correctly. Source tree. readopr2 is GATEC-specific.

5. \$CVSROOT/wang depends on readopr2. make install installs the following binaries

into LOCALBIN:

BCAScancelaward - Gateway script to upload canceled purchase orders to BCAS.

BCASitemupload - Gateway script to upload item description changes to BCAS.

BCASrunproc - Gateway script to run a WANG procedure.

BCASupload - Gateway script to upload awards (purchase orders).

cancelerrtouser - Bourne shell script used to send cancel award errors to specified users.

cdf\_check - Perl script that uses the regular expressions defined in /home/gatec2/etc/cdf\_regex to grammatically check content of award upload, item upload, and cancel award upload CDFs.

downloadch - Perl script used by getwangfiles to ftp files from WANG to a UNIX machine.

errtomgr - Bourne shell script used by \_cron files to send errors to the gatecmgr.

errtouser - Bourne Shell script used by \_cron files to send error information to specified users.

get\_UTNNumber\_from\_cdf - Perl script used by putuploads\_cron to obtain UTN number from award CDF.

get\_piin\_from\_cdf - Perl script used by putuploads\_cron to obtain piin number from upload CDF

getopr\_bsp\_cron3 - Bourne shell script run as cron to download new RFQs into GATEC database (makes use of getwangfiles and BCASrunproc).

getstmtntship\_cron - Bourne shell script run as cron to load Shipping, Account, and Statement tables (makes use of getwangfiles and BCASrunproc).

getwangfiles - Bourne shell script used by getopr\_bsp\_cron3 and getstmtntship\_cron to ftp BCAS data from WANG to SPARC database machine

iiitemerrtouser - Bourne shell script used to send item upload error information to specified users.



putuploads\_cron - Bourne shell script run as cron to upload award information to BCAS. Makes use of BCASupload, seluploadcdf and the "C" queue routines (see below).

selcancelcdf - Perl script which writes all non blank fields in a cancel CDF to standard out.

selitemcdf - Perl script which writes all non blank fields in an item CDF to standard out.

seluploadcdf - Perl script which writes all non blank fields in an upload CDF to standard out.

setUTN\_aw\_to\_cl - PL/SQL script used by putuploads\_cron to change a state of an acquisition from AWARDED to CLOSED.

setUTN\_ignoreack - PL/SQL script used by putuploads\_cron to tell database to ignore acknowledgment checking.

upload\_bcas\_cancel\_award\_cron - Bourne shell script run as cron that uploads cancel award information to BCAS. Makes use of BCAScancelaward, and selcancelcdf.

upload\_bcas\_item\_desc\_cron - Bourne shell script run as cron that uploads item description information to BCAS. Makes use of BCASitemupload and selitemcdf.

#### Other Miscellaneous Executables

close - Bourne shell script to close all OPEN RFQs.

daily - Bourne shell script used to close RFQ's in GATEC database after they have been open for a designated period of time. It is run as a cron

downloadPiins - Bourne shell script used to add new award piins to GATEC database.

downloadbsp - Bourne shell script used to add new buyers to GATEC database.

get\_login\_info - Uses ASCII file (login) in /home/gatec2/etc to obtain password for access to ORACLE/GATEC database under SQLPLUS.

sh\_get\_login\_info - Uses ASCII file (login) in /home/gatec2/etc to obtain password for access to

ORACLE/GATEC database under SQLPLUS.

insertHolidays - Bourne shell script used to add Holiday information to GATEC database

Note cdf\_regex is installed into LOCALETC. Seven man pages are installed into LOCALMAN. There is one minimal regression test. To run it, su to gatecmgr, and type "make putupload\_cron\_test".

All the wang code is GATEC-specific.

6. \$CVSROOT/v8regex make install creates libv8regex.a. libv8regex.a is installed into LOCALLIB. v8regex.h is installed into LOCALINC v8regex.3 is installed into LOCALMAN. \$CVSROOT/v8regex is not GATEC-specific and can be re-used.

---

## 1.11 Miscellaneous Software

---

---

### 1.11.1 Acknowledgment Monitoring Software

---

#### NAME

ack\_cron\_pl - Look for 997 rejects and overdue 997s for X12 documents sent by the site.

#### SYNOPSIS

ack\_cron\_pl

#### DESCRIPTION

ack\_cron\_pl creates a list of 997s received for the current date that report a rejection of a document the site has sent. It also creates a list of X12 transactions the site has sent which have not been acknowledged within the period of time defined in the table UserManagerDefaults.

This list is mailed to the address stored in the NotificationAddress column of the UserManagerDefaults table. In addition, notification is sent to each buyer with the acqerr program.

Finally, the DocumentSent table is updated to indicate that

rejection warning or an overdue warning has been sent.

This program is typically run from cron, once a day, when no users are on the system. It can take up to 30 minutes to run.

## TABLES

UserManagerDefaults	Contains email address of where to send report
s_SendRejectWarning	select-only view on DocumentSent, FunctionalAck, Document
s_SendOverdueWarning	select-only view on Document, DocumentSent, UserManagerDefaults, FunctionalAck
u_SendRejectWarning	update-only view on DocumentSent, FunctionalAck
u_SendOverdueWarning	update-only view on DocumentSent, FunctionalAck, UserManagerDefaults

## SEE ALSO

acqerr(1), sh\_get\_login\_info(1)

## NAME

997CDFtoDB - insert a 997 X12 document into the database.

## SYNOPSIS

997CDFtoDB

## DESCRIPTION

997CDFtoDB reads from stdin. It reads a cdf file of type 997. Typically, the file is generated by the Translator.

997CDFtoDB verifies that the cdf file is syntactically correct. If so, it inserts the data into the database.

997CDFtoDB exits 0 if the data was successfully inserted, exits 1 if there is a syntax error in the cdf, and exits 2 if the database is unavailable.

## DATABASE TABLES

Document  
FunctionalAck

SEE ALSO

v8regex(3), Connection(3N), Database(3N), Table(3N),  
get\_login\_info(3), outline\_wade(1)

## Compilation/Installation of Acknowledgment Monitoring Software

6. \$CVSROOT/arc depends on an active, functioning database.  
Make install installs ack\_cron\_pl into LOCALBIN.

There are regression tests here. Run them with "make test". These tests can take over an hour if there are lots of 997's already in the database.

ack\_cron\_pl is GATEC-specific.

See ack\_cron\_pl(1).

ack\_cron\_pl - Perl script used to look for newly rejected and overdue documents.

acqerr - used by ack\_cron and putuploads\_cron to associate an error message with an acquisition in the GATEC database.

Note: arc also contains a data model, using Rumbaugh's Object Modeling Technique. This model sketches an alternative implementation of X12 using a relational database. See arc/doc, arc/schema, and arc/src.

9. \$CVSROOT/dui/src/cdfdb/997CDFtoDB depends on v8regex, narq, and nora. xmkmf make Makefile make depend make install997 installs 997CDFtoDB into LOCALBIN.

There is a regression test. To run it, type "make test997"

See 997CDFtoDB(1). An equivalent implementation of this

Sun Release 4.1

Last change:

2

OUTLINE\_WADE(1)

USER COMMANDS

OUTLINE\_WADE(1)

program is oci997CDFtoDB.cc.

\*\*\*\*\* Testing \*\*\*\*\*

Ideally, every Makefile or Imakefile should have a test: target, such that % make test would cause a reasonably complete regression test to be run on all modules referenced in the Makefile.

In practice, readopr2, arc, and 997CDFtoDB are the only GATEC modules I know of with a "% make test" regression test. There may be others.

SEE ALSO

GATEC(1), db(3), q(3), v8regex(3), getopr\_bsp\_cron3(1), putuploads\_cron(1), readopr2(1), 997CDFtoDB(1), ack\_cron\_pl(1)



---

---

## SECTION 2 Distributed User Interface (DUI) Software

---

The Distributed User Interface Software (located at \$CVSROOT/dui in the GATEC development environment) includes the DUI toolkit used by both client and application to communicate objects between the PC and SPARC machine, windui application which implements display of these objects on a PC running Windows 3.1, and the GATEC application itself. The Lead Buyer and System Parameter client applications (like the GATEC application) also use DUI. They are described in this section as well

---

### 2.1 DUI Toolkit

---

DUI is a client-server system for building platform- independent user interfaces. It allows an application programmer to write a user interface that can be displayed in any environment for which a DUI client has been written. It also allows the application and user interface to run on separate machines thereby distributing the user interface processing.

It was designed to separate the I/O needs of the application from the display and formatting requirements of the user interface. This greatly simplifies the task of the application programmer. It does this by providing a set of simple tools describing the basic forms of input and output an application requires to interact with a user. Some of the basic tools are:

- Form

a "screen" which will contain any number of the other tools as well as a set of Command's that would operate on the data described in the form.

- Command

a way for the user to act on the data entered or selected on the form.

- Selection

a list of items from which the user can select one.

- Multi\_Selection  
a list of items from which the user can select more than one.

- Toggle  
a switch that can be turned either on or off.

- Field  
a data entry field in which the user can enter one line of data.

- Text  
a data entry field in which the user can enter more than one line of data.

- Range  
a range of values from which the user can select one.

- Table  
a data entry tool that allows the user to edit rows and columns in a tabular format.

These tools are used by the application and implemented by a "client". The client is a program that implements these tools for a particular display environment. It is written once and can handle any application that uses DUI. By separating it in this way the ability for a single application to run on multiple display platforms is provided.

---

### 2.1.1 Basic Architecture

---

The DUI system consists of three parts - an application, a server, and a client. These three pieces function in the following ways:

#### Application

This is the piece that is written by a developer to perform whatever its requirements specify (one of which is to provide a forms type interface to an end user). It makes calls to the two DUI libraries libdui\_comm and libduit (see the CLASS HIERARCHY AND LIBRARIES Section) in order to build its interface. It is executed by the "server" process on request from the "client". Its executable name always ends in ".dui". See the APPLICATION PROGRAMMING GUIDE section for more about DUI application programming.

---

#### 2.1.1.1 Server

---



This is the executable that the client talks to at initial startup once a communications pathway has been established. It resides on the same machine as the application and responds to a request from the client to start up an application. The client supplies the application executable name (minus the ".dui") and a search path. Once the application is started up the server terminates and the application and client communicate directly (see COMMUNICATIONS Section).

---

#### 2.1.1.2 Client

---

This is the program that implements the toolkit elements for a particular platform (e.g. XWindows, MS Windows, Macintosh). It also makes use of the two DUI libraries but must port and make extensions to them (see CODE GENERATION section) to establish an appropriate communications path and implement the display specifics for its platform. It initiates the communications link and requests a particular application (by sending a control object) (see COMMUNICATIONS Section). It then displays forms (sent by the application) to the user and communicates the user interaction back to the application. It does this until either side elects to terminate.

---

#### 2.1.2 Communications

---

The basis of the DUI system communications is the ability for the client and the application to communicate shared objects across a communications link. This involves establishing a link between the client and the application and providing a "protocol" for communicating "objects" back and forth between the two processes. These are done as follows:

---

##### 2.1.2.1 Communications link

---

DUI communications is done through C++ streams. There is a class, Session (1), which is responsible for opening the streams. It contains one input and one output stream. The client, application, and server each contain one instance of a Session. The input and output streams for the application and server are currently always standard in and standard out respectively. The client on the other hand must establish its input and output streams by whatever

communications channel is appropriate (e.g. sockets or serial line) to make a connection to the server and attach to its standard input and standard output. Once the client has established a connection with the server an AppControl (1) object is passed to the server to request a particular application, the server executes the named application giving it its standard in and standard out and terminates. This leaves the application and client communicating directly. See the DETAILED WORKING EXAMPLE section for an example of establishing a connection between a client and a server.

---

#### 2.1.2.2      DUI Protocol

---

The idea behind the DUI protocol is the sharing of class instances by the client, server and application. So the protocol is basically an ASCII representation of class instances and instance hierarchies. The application, server and client simply send instance hierarchies and modifications to individual instances in those hierarchies back and forth to each other to maintain duplicate copies. Not all classes are communicated, only those that describe user interface elements. Each class that needs to be communicated can write its data member onto a stream in a form which identifies its class, instance and data. By the same token it can read itself in. There is a class called Communication\_Object (1) which is the base for all classes that are communicated. Derived classes overload some of its functions to write out and read in their specific data elements. Each process keeps a global list of every instance created by either side so modifications can be applied to the right instance. It does this by overloading the new operator for all communicable classes. In addition class structures on the client side can be different from those on the application side in that the client can add more data and function members to the original DUI class definition in order to implement its display functionality. This is done without subclassing (See the CODE GENERATION Section). These additional data members however are local to the client and are not communicated to the application. The application can add local (non communicated) data members by sub-classing off of the DUI classes (See the DETAILED WORKING EXAMPLE and the APPLICATION PROGRAMMING GUIDE Sections for more details).

---

#### 2.1.2.3      A Typical Session

---

The following is a short summary of what a typical

communications session is like for a DUI application:

The client starts. The client establishes a link to the server. The client sends an AppControl (1) object to the server. The server finds the named application and terminates leaving the application and client communicating directly. The application creates a DUI\_Form(1) attaching other widgets to it (e.g. DUI\_Field's, and DUI\_Command's) It then sends this instance hierarchy to the client to be displayed. The client displays the form. The user interacts with the form. The client communicates the modifications the user made to the form back to the application. The application acts on the modifications possibly sending back another form. This continues until either the client or the application sends an AppControl object back signaling termination. Finally, both the client and the application terminate. For an in depth example of client-application interaction see the DETAILED WORKING EXAMPLE Section.

---

### 2.1.3 DUI Class Hierarchy and Libraries

---

DUI consists of two class libraries, libdui\_comm and libdui, which are written in C++. The libraries and the classes they contain are described below. In the class hierarchy descriptions indentation denotes derivation (i.e. In the first diagram DUI\_Form is derived from DUI\_View which is derived from DUI\_Widget which is derived from Communication\_Object). For in depth explanations of each of the classes see their individual documentation, and for an explanation of their usage see the APPLICATION PROGRAMMING GUIDE Section.

Libdui contains the DUI toolkit elements. These are the classes that define Fields, Views, Commands etc.(i.e. user interface widgets). They are all derived from DUI\_Widget (1) which itself is derived from Communication\_Object (1). All of these classes can be modified by the client using special include files (see CODE GENERATION SECTION) which is the reason they are all contained in one library. The following is the class hierarchy for libdui (except Communication\_Object which is contained in libdui\_comm):

```
(Communication_Object)
  DUI_Widget
    DUI_View
      DUI_Form
      DUI_Dialog
    DUI_Component
      DUI_Command
```

- DUI\_End\_Command
  - DUI\_Toggle
  - DUI\_Range
  - DUI\_Field
- DUI\_Invisible\_Field
  - DUI\_Text
  - DUI\_Group
  - DUI\_Label
  - DUI\_Selection
- DUI\_Multi\_Selection
  - DUI\_Table

Libdui\_comm contains all the rest of the classes that are used in the DUI system which includes the classes used for communications. Broken down by category they are:

Classes used for communications:

- Filebuf\_With\_Audit
- Session
- ChannelBuf
- SocketBuf
- ConfigInfo
- Communication\_Object
- AppControl

The next category describes Modifiers and Constraints these are communicable classes that can be attached to DUI\_Fields, DUI\_Texts, and Table\_Columns. Modifiers and Constraints are applied to the contents of their associated widgets after the user has entered something. The modifiers allow the application programmer to automatically change the contents before the application receives the value (e.g. upcase the contents) and constraints provide a way for the application to enforce a format (i.e. Integer) on the contents. See Modifier (1), Constraint (1) and the APPLICATION PROGRAMMING GUIDE section for more information about their use.

Modifiers and Constraints:

- Communication\_Object
- Modifier
  - Justified
  - Left\_Justified
  - Lower\_Case
  - Precision
  - Right\_Justified
  - Truncated
  - Unjustified
  - Upper\_Case

Constraint  
Date  
Integer  
Mandatory  
Military\_Date  
Numeric  
Regular\_Expression

The two remaining classes are utility classes. Table\_Column defines the functionality for columns in a DUI\_Table widget and STRING provides a generic string class. They are both communicable classes.

Communication\_Object  
Table\_Column  
STRING

---

#### 2.1.4 Application Programming Guide

---

The following are things an application programmer needs to know to create a DUI Application. See DETAILED WORKING EXAMPLE for an example of a DUI application.

---

##### 2.1.4.1 Beginning and Ending a session

---

The application programmer must call Session::begin() at the beginning of her program and Session::end() at the end. These functions setup and shut down the input and output streams and send AppControl objects to confirm startup and initiate termination.

---

##### 2.1.4.2 Event Driven Programming

---

DUI Applications are event-driven. Forms are created that contain components with functions attached to them. These functions are callback function (see below). When a user modifies or elects a particular component these functions are activated. The only "event" is the modification of a component, so the contents of the component should determine the action taken. (see Call Back Functions below).

The application should sub-class off of the `DUI_Form` class to create the forms it requires. `DUI_Form` has one `DUI_Component` member and one `DUI_Command` member. If the form requires more than one component (which most do) then they should be grouped under a `DUI_Group` and this should be used to set the forms `DUI_Component` member. If the form needs more than one command (which again most do) then group the `DUI_Command`'s under another `DUI_Command` (`DUI_Command`'s can be used to group other `DUI_Command`s) and set this top level `DUI_Command` to the form's `DUI_Command` component. See `DUI_Form(1)` and `DUI_View(1)` for more information about the structure of a form. The components which can be attached to `DUI_Form`'s are as follows:

`DUI_Field`

This is a single line data entry field.

`DUI_Group`

This is a component which can contain other components. So this is used to logically group other components. It can contain `DUI_Group`'s so it is recursive.

`DUI_Toggle`

This is a switch that can be turned on or off by the user.

`DUI_Label`

This is read only text.

`DUI_Multi_Selection`

This is a list of item from which the user can select more than one.

`DUI_Selection`

This is a list of item from which the user can select one.

`DUI_Text`

This is multiple line data entry field.

`DUI_Range`

This is a range of numeric values from which the user can pick one.

`DUI_Command`

This is an action the user can take. It has callback (see below) functions associated with it that are executed on the application

side when the user elects them.

#### DUI\_End\_Command

This is the same as DUI\_Command except that it cause the view on which it appears to go away after the user elects it.

#### DUI\_Table

This is a complex widget that allows a user to edit rows and columns in a tabular format. Its functions include adding and deleting rows and changing the contents of columns. It uses the utility class Table\_Column(1) to handle column functions.

#### DUI\_Dialog

This can not be attached to a form but is a stand alone form itself. It is used primarily for simple dialogs (e.g. confirmation and error messages), but can support more complex forms.

---

### 2.1.4.4 Modifiers and Constraints

---

Modifiers and constraints can be attached to DUI\_Field's, DUI\_Text's and Table\_Column's. They are used to automatically modify and validate the contents of these components. Modifiers are used if you want to change the format of the field but don't need to make sure the user enters it that way. The modifiers are:

Justified, Left\_Justified, Lower\_Case, Precision, Right\_Justified Truncated, Unjustified, Upper\_Case.

Constraints are used if the programmer needs to ensure that the user enters data in a specific format the error processing is done on the client side before it gets back to the application, and DUI\_Fields and Table\_Columns will not allow their values to be set to something that does not conform to their constraints. The Constraints are:

Date, Integer, Mandatory, Military\_Date, Numeric, Regular\_Expression

---

### 2.1.4.5 Callback Functions

---

Callback functions can be assigned to any of the descendants of DUI\_Component. A callback function is executed whenever a modification to the component it refers to is received from the client. In this way the case where the contents of one component

affects the contents of another can be handled. Usually `DUI_Command`'s are always given callbacks since they represent actions the user can take, but there are cases where it is not necessary. For Instance, if you want a command to simply end the screen you are on, you would use a `DUI_End_Command` and not give it a callback function. There are many examples of instances where callback functions would be useful on other type of widgets, one being exclusive toggles (where only one toggle can be selected out of a group).



---

### 2.1.5 Code Generation

---

A significant portion of the code for the DUI libraries and DUI clients are generated from formatted descriptions of the classes. The tools used to generate the code are lex and yacc parsers and c++ programs. There are two different categories of generation and they are described below.

Code generation for communicable classes For all communicable classes, the functions `co_print()`, `co_parse()` (i.e. the functions that write out and read in a class' data members), the class declaration (contained in its header file) and stubs for unwritten member functions (contained in a .C file if one does not exist for the class) are generated using a tool called "expand\_class". Expand\_class reads a description file named "<class name>.def" and produces three files called "<class name>.C", "<class name>.gC" and "<class name>.H". The ".C" file is created only if one does not exist. The ".H" file contains the class declaration with place holders for client additions (see below), macros for overloading the new and delete operators (which keep track of the global list of instances) and forward declarations and include files if the class requires any. The ".gC" file contains the print and parse routines (`co_print()` and `co_parse()`) and is included into the ".C" file. Another utility called "make\_find" generates a file called "DUI\_find.C" which contains functions for finding an instance of a class given its numeric class and instance ids (or creating a new instance if it does not exist) and retrieving a class' numeric id.

---

#### 2.1.5.1 Client Code Generation

---

When a new client is created the entire DUI\_Widget class tree is copied and the "DUI\_" prefix for all the classes is changed to "<client prefix>\_" (e.g. "DUI\_Field" is changed "w\_Field" for a Windows client). It is then given the opportunity to make changes directly to the DUI interface classes using three types of include files which are incorporated into the class declaration when generated (see above). The files are "<client prefix>\_decls.HH", "<client prefix>\_<class name without DUI prefix>.HH", and "<client prefix>\_<class name without DUI prefix>.CC". Additions are made to the ".HH" and ".CC" as if they would occur inside a class declaration and member function source file respectively. The "<client prefix>\_decls.HH" is incorporated into every class so is designed for class additions that affect all the classes. More than this may need to be done to port the library code depending on the clients environment, but this is provided to make writing clients easier.

The DUI source directory includes not only the two dui libraries but the clients and applications written for it. The directory has the following structure (indentation denotes a sub directory):

```
$CVSROOT/dui/  
bin/  
etc/  
include/  
    dui/  
    duimake/  
lib/  
src/  
    applications/  
    clients/  
    generate/  
    libdui_comm/  
    libduit/
```

The source for the two libraries are kept in \$CVSROOT/dui/src/libduit and \$CVSROOT/dui/src/libdui\_comm. Imake is used as the make utility so to make everything in the directory structure which includes the two libraries go to \$CVSROOT/dui, and type:

```
xmkmf; make World
```

It will descend through the directories and attempt to make everything in them. It will fail to make the applications if the other libraries they depend on are not made and installed where their make files expect to find them.

Most of the include files in \$CVSROOT/dui/include/dui are links to the files in the two library source directories. The linking is done by make World. The files must be linked here in order for the two libraries to make. If it is necessary to make the libraries by themselves, then link the header files into this directory, cd to the library source directory and type:

```
xmkmf; make predepend depend all
```

This can be done in the two library source directories.

The libraries get installed in \$CVSROOT/dui/lib whether they are made individually or with make World. The application binaries if they were made get installed in \$CVSROOT/dui/bin, as well as the clients that are intended for the machine the make is being done on if any.

See individual client and application documentation for how they are integrated into the dui source if at all.

The tools for generating the code are kept in the \$CVSROOT/dui/src/generate. These are made automatically by the library make files if they are not made yet or out of date. The resulting binaries are left and accessed in their source directory.

---

### 2.1.7 Detailed Working Example

---

The following is the source and a detailed description of a typical session for a simple DUI application. The application resides on a UNIX machine and provides a form for reading a disk file that the user requests by entering a file name in a field provided on the form. It then displays the contents of this file to the user. The display environment is a PC running Windows 3.1 that has a modem that can dial in to a tty on the UNIX machine.

The client is run with a command line of:

```
windui.exe file_reader
```

with the following in its DOS environment

```
SET APP_PATH=/public/app:/public/new_app
```

The following executable programs are on the UNIX machine:

```
/public/new_app/file_reader.dui    /home/duiuser/dui_shell  
/home/duiuser/dui_server
```

and a text file:

```
/home/duiuser/text.file
```

The Windows client is started up. On its command line is the name of the application to run on the remote machine ("file\_reader"), and in its environment a variable is defined containing the search path to use on the remote machine ("/public/app:/public/new\_app"). It negotiates a modem connection to the UNIX machine and logs in as a public DUI user. This is done with a serial communications script on the PC. The public DUI user has the following program as its login shell /home/duiuser/dui\_shell. The shell sets the tty for raw io and executes the /home/duiuser/dui\_server program which is the dui "server". The client then sends an AppControl Object containing the name and path. The server executes the program looking in the supplied paths and terminates. The file\_reader

application sends a `DUI_Form` containing a `DUI_Field` and a `DUI_Text` (grouped under a `DUI_Group`) and two `DUI_Command`'s grouped under another `DUI_Command` (`DUI_Command`'s can act as groups for other `DUI_Command`'s). This form is displayed by the Windows client as Window containing a edit field with the title "File Name", a multi-line edit field with the title "Contents" and two buttons named "Open" and "Quit". The user enters `"/home/duiuser/text.file"` in the "File Name" field and presses the "Open" button. As soon as the user exits the "File Name" field the `DUI_Field` is changed on the client side and the change is communicated to the application. When the user presses the "Open" button the `DUI_Form` is sent back to the application where the callback function, `"read_file()"`, attached to that button is executed. `Read_file()` reads the file and puts its contents into the `DUI_Text`. This causes an update to be sent back to the client which updates the window to reflect the new contents of the `DUI_Text`. After the user views the file he presses the "Quit" button. The Form is sent back to the application where the callback `"quit()"` is executed. This function sends an `AppControl` object back to the client through the `Session::end()` function to end the session. The client receives the `AppControl` object and terminates. The application then terminates.

If you look at the following source, you will notice in the file called `File.C` there is the main function for the application. It sets up the session using `Session::begin()` (which opens the streams), instantiates a new `File_Form` and displays it. The definition for `File_Form` is found in `File_Form.C` and `File_Form.H`. It is derived from `DUI_Form` and in its constructor it sets the `component()` and `command()` members for the form to its `DUI_Group` which contains a `DUI_Field` and a `DUI_Text` and its `DUI_Command` which contains its two other commands. When the `DUI_Command`s are instantiated they are passed callback functions which are the other member functions for `File_Form`. This is all the source required to write this application.

#### `File.C`

```
/*
 * main() for File application
 */

#include <stdlib.h>
#include <dui/DUI.H>
#include <dui/Session.H>
#include "File_Form.H"

main( int argc, char **argv )
{
    if ( Application_Session::begin(argv[0]) == -1 ) {
        Session::log() << "Session::begin failed." << endl;
    }
}
```

```

exit(-1);
}

new File_Form()->display();

exit(-1);
}

-----

File_Form.H
#ifndef File_Form_HEADER
#define File_Form_HEADER

/* File_Form.H
*/

#include <dui/DUI_Field.H>
#include <dui/DUI_Text.H>
#include <dui/DUI_Form.H>

class File_Form : public DUI_Form {
public:
    File_Form();
    ~File_Form();

    // Callback functions

    void quit();
    void read_file();

    // important Form components

    DUI_Field * file_name_;
    DUI_Text * file_contents_;
};

#endif

-----

File_Form.C
//
// Methods for File_Form Class
//
static const char rcsid[] = "$Id:$";

#include <fstream.h>
#include <stdlib.h>
#include <unistd.h>

```

```

#include <errno.h>

#include <dui/DUI.H>
#include <dui/Callback.H>
#include <dui/Session.H>
#include <dui/DUI_Group.H>
#include <dui/DUI_Text.H>
#include <dui/DUI_Field.H>
#include <dui/DUI_Command.H>
#include <dui/DUI_End_Command.H>
#include "File_Form.H"

declare(Callback_Function,File_Form)
implement(Callback_Function,File_Form)

/* Constructor for File_Form
*/
File_Form::File_Form() :
DUI_Form("View File")
{

    // Create Form and callbacks

    // Create the data entry field that will hold the
    // file name and the text field that will hold the
    // contents of the file.
    // Group them and assign the group to this Form's
    // "component" member.
    component(
    new DUI_Group( "View File",
file_name_ = new DUI_Field("File Name"),
file_contents_ = new DUI_Text("Contents")
    )
    );

    // Create commands for opening a file
    // and quitting the application,
    // create callback functions to executed
    // when command is pressed,
    // group them, and assign the group to
    // the Form's "command" member.
    command(
    new DUI_Command( "",
    new DUI_Command( this, "Open" ,
    new Callback_Function(File_Form)(
    this, &File_Form::read_file)
    ),
    new DUI_End_Command( this, "Quit" ,
    new Callback_Function(File_Form)(
    this, &File_Form::quit)

```

```

    )
    )
};
}

File_Form::~File_Form()
{
    delete file_name_;
    delete file_contents_;
}

// Checks to make sure file is accessible for reading,
// if so reads the first
// ten lines and displays them.

void
File_Form::read_file()
{
    file_contents_->reset_line_count();
    STRING err_msg("");
    if (access(file_name_->value(), R_OK) != 0) {
if (errno <= sys_nerr) {
    err_msg += sys_errlist[errno];
} else {
    err_msg += "Access returned unknown file error.";
}
file_contents_->append_line(err_msg);
return;
    }
    STRING file_lines("");
    int line_count = 0;
    ifstream current_file(file_name_->value());
    while (!current_file.eof() && line_count++ <= 20) {
char file_line[256];
current_file.getline(file_line, sizeof(file_line)-1);
file_lines += "0;
file_lines += file_line;
    }
    file_contents_->lines(0, file_lines);
    return;
}

// Calls Session::end() to end this DUIT Session.

void
File_Form::quit()
{
    Session::end();
}

```

---

## 2.1.8      DUI Detail Class/Object Descriptions

---

The classes/objects which are used to implement the DUI toolkit are listed below:

AppControl  
ChannelBuf  
Communication\_Object  
ConfigInfo  
Constraint  
DUI\_Command  
DUI\_Component  
DUI\_Dialog  
DUI\_End\_Command  
DUI\_Field  
DUI\_Form  
DUI\_Group  
DUI\_Invisible\_Field  
DUI\_Label  
DUI\_Multi\_Selection  
DUI\_Range  
DUI\_Selection  
DUI\_Table  
DUI\_Text  
DUI\_Toggle  
DUI\_View  
DUI\_Widget  
Date  
Filebuf\_With\_Audit  
Integer  
Justified  
Left\_Justified  
Lower\_Case  
Mandatory  
Military\_Date  
Modifier  
Numeric  
Precision  
Regular\_Expression  
Right\_Justified  
STRING  
Session  
SocketBuf  
Table\_Column  
Truncated  
Unjustified  
Upper\_Case



These items are described in the following pages

### 2.1.8.1 AppControl

#### NAME

AppControl - Used to pass control information between application and client.

#### SYNOPSIS

```
#include "AppControl.H"

class AppControl: public Communication_Object {

communication_decls(AppControl)
protected:
    int _end;
    STRING*  appname;
    STRING*  apppath;
    void receive();
public:
    void(*exitfp_());
    friend Session;
    AppControl();
    AppControl(const char *name, void(*exitfp)());
    ~AppControl();
    int execute();
    int end();
    int end(int newend);
    const char *name();
    virtual short updated() const { return 1; };
    virtual short need_to_update() const { return 1; };
public:
    virtual const char *class_name() const { return "AppControl"; }
}
```

#### DESCRIPTION

This class is used to pass the application name and search path from the client to the dui server (a specialized application). The server uses this information to start up an application. It is also used by either the application or the client to tell the other that it is time to shutdown.

#### MEMBER FUNCTIONS

AppControl::AppControl (const char \*name, void(\*exitfp)

Description: Constructor accepting an application name and exit function pointer. The exit function is called when this object is

received and the end flag is set. returns: void

AppControl::~~AppControl()

Description: Destructor. Deletes application name and search path.  
returns: void

AppControl::AppControl()

Description: Empty constructor. returns: void

int AppControl::end()

Description: Assessor function. returns: int end flag.

int AppControl::end(int newend)

Description: Sets end flag. returns: int new end flag.

int AppControl::execute()

Description: This function attempts to execute the application named by its appname member plus a ".dui" extension using the search path specified in its apppath member. It appends the paths "." and "./appdir" to the end of the search path before executing, using execlp(). It does not fork. returns: int -1 if the exec failed otherwise it does not return.

void AppControl::receive()

Description: The receive function for this Communication\_Object. It check the end flag and calls the exit function and exits if it is set.  
returns:  
void

const char \*AppControl::name()

Description: Assessor function. returns: const char \* the application name.

## FILES

AppControl.C AppControl.H

### 2.1.8.2 ChannelBuf

#### NAME

ChannelBuf - Base class for DUI streambuf's.

#### SYNOPSIS

```
#include "ChannelBuf.H"
```

```
class ChannelBuf: public streambuf { public:  
    friend class Session;  
    friend class Channel;  
    ChannelBuf();  
    virtual ~ChannelBuf();  
private:  
    int state;  
    int inerror();  
    virtual int connect();  
    virtual int disconnect();  
}
```

#### DESCRIPTION

This class is a base class for the types of streambufs DUI uses for communications. It provides a status field and dummy functions for connect() and disconnect().

#### MEMBER FUNCTIONS

ChannelBuf::~~ChannelBuf()

Description: Destructor. Calls disconnect. returns:  
void

Description:

Empty Constructor. returns: void

int ChannelBuf::inerror()

Description: Accesser function. returns: 1 if error and 0 otherwise.

int ChannelBuf::connect()

Description: Place holder function for derived classes to overload.  
returns: 0 always.

int ChannelBuf::disconnect()

Description: Place holder function for derived classes to overload.  
returns: 0 always.

#### FILES

ChannelBuf.C ChannelBuf.H

### 2.1.8.3 Communication\_Object

#### NAME

Communication\_Object - Base class for all objects that must be communicated.

#### SYNOPSIS

```
#include "Communication_Object.H"

class Communication_Object { private:
    short updated_;
    short need_to_update_;
    static short update_ok_;
protected:
    long oid_;
    short is_pointer_;
    virtual void check_pointer() {}
    short updates_ok() { return update_ok_; }
    void updates_ok( short ok ) { update_ok_ = ok; }
    Communication_Object();
    virtual void send();
protected:
    friend class Session;
    virtual void receive();
protected:
    friend ostream &operator <<( ostream &, Communication_Object * );
    virtual void co_print( ostream & );
    virtual void co_parse( istream & );
    void need_to_update( short n ) { need_to_update_ = n;
}
public:
    virtual int class_id() const { return 0; }
    virtual ~Communication_Object();
    static Communication_Object *read_in(istream & );
    long oid() const { return oid_; }
    virtual short updated() const { return updated_; }
    virtual short need_to_update() const { return need_to_update_; }
    virtual void update( int changed = 1 );
}
```

#### DESCRIPTION

This class provides a base for all classes that must be communicated between the application and the client. For a discussion of the DUI communications paradigm see DUI.

#### MEMBER FUNCTIONS

)

ostream & operator << ( ostream & out, Communication\_Object \*obj  
Description: This function overloads the << operator for Communication\_Object. It checks the object it is attempting to write out to see if it needs to be sent in full (is updated) if not it just writes out the class id and object id (i.e. stubs) of the object. This is obviously done for efficiency. returns: ostream & "out".

Communication\_Object::read\_in( istream &in )

Description: Determines the type of the object described by "in" and parses the object. All objects are given a class id and an object id (identifying a particular instance). With this information an appropriate object is either created and given the data that is on the stream or if the object has already been created finds the object in the active object list and updates its data elements with what is on the stream. returns: Communication\_Object \* the object read in.

Communication\_Object::Communication\_Object()

Description: Empty constructor. returns: void

Communication\_Object::~~Communication\_Object()

Description: Destructor. Does nothing. returns: void

void Communication\_Object::receive()

Description: This function is called on each object which is received, and is overloaded where appropriate. returns: void

void Communication\_Object::send()

Description: Sends object using Session::send() (which see). returns: void

void Communication\_Object::update( int changed )

Description: Send update of object. This function is called by functions in derived classes which modify the data members there by causing them to be retransmitted. There is an update\_ok\_ flag which is checked before sending to see if it is desirable to actually transmit the data. This is there so that repetitive operations that change the data can be performed without causing re-transmission on each repetition which can be expensive. returns: void

Communication\_Object::co\_print( ostream &out )

Description: This function writes out the Communication\_Object class' data members. Namely:  
class\_id and oid. returns: void

Communication\_Object::co\_parse( istream & )

Description: This function does nothing and is a place holder for derived classes. One would expect it to read in the class' data members but those are read in by read\_in() to determine which object is on the stream. returns: void

## FILES

Communication\_Object.C Communication\_Object.H

### 2.1.8.4 Constraint

## NAME

Constraint - Base class for DUI constraints.

## SYNOPSIS

```
#include "Constraint.H"
```

```
class Constraint: public Communication_Object {  
  
    communication_decls(Constraint)  
    public:  
        virtual ~Constraint();  
        virtual const char *invalid(const char *string) const;  
    protected:  
        Constraint();  
    public:  
        virtual const char *class_name() const { return "Constraint"; }  
}
```

## DESCRIPTION

This class is the base class for more specific DUI constraints such as Integer(which see). Constraints can be applied to DUI\_Field's(which see), and Table\_Column's(which see). They provide a way for the application programmer to constrain the format of a value. Constraints are checked whenever the value is being set for the element the constraint is assigned to. If the constraints are not met the value is not assigned. It is intended that the client warn the user of the violation of a constraint and allow him to enter another value. In this way the application programmer knows that a value will have a certain format without checking.

## MEMBER FUNCTIONS

Constraint::Constraint()

Description: Empty constructor. returns: void

Constraint::~~Constraint()

Description: Destructor. Does nothing. returns: void

const char \*Constraint::invalid(const char \*)

Description: This function must be overloaded by derived classes. It is intended to check the passed value against the rules of the constraint and return an error message if it fails. returns: "" always.

## FILES

Constraint.C Constraint.H  
**2.1.8.5 ConfigInfo**

## NAME

ConfigInfo - A class for storing the communications configuration information.

## SYNOPSIS

```
#include "ConfigInfo.H"
```

```
class ConfigInfo { private:
```

```
    int status;
```

```
    char *apphost;
```

```
    int sport_;
```

```
    char *comport_;
```

```
    int baud_rate;
```

```
    int data_bits;
```

```
    int stop_bits;
```

```
    char *parity_;
```

```
    char *type_;
```

```
    char *connect_script;
```

```
    char *server_;
```

```
    char *logon_;
```

```
public:
```

```
    ConfigInfo();
```

```
    ConfigInfo(char *atype, char *aapphost, int asport, char *acomport, int abaud_rate, int  
adata_bits, int astop_bits, char *aparity, char *aserver);
```

```
    ~ConfigInfo();
```

```
    char *host();
```

```
    char *type();
```

```
    char *parity();
```

```
    int sport();
```

```
    char *comport();
```

```
    int baud();
```

```
    int data();
```

```
    int stop();
```

```
    char *script();
```

```
    char *server();
```

```
    char *logon();
```

```
    int inerror();
```

```
    char *host(char *);
```

```
    char *type(char *);
```

```
    char *parity(char *);
```

```
    int sport(int);
```

```
    char *comport(char *);
```

```

int baud(int);
int data(int);
int stop(int);
char *script(char *);
char *server(char *);
char *logon(char *);
int write();
}

```

## DESCRIPTION

This class reads from a file called "com.cfg" which it expects to find in the current directory, and fills in its data members. It expects com.cfg to be in one of the following formats:

serial <comport> <baudrate> <parity> <databits> <stopbits> <logon> <script>

socket <hostname> <port>

epipes <server prog name>

## MEMBER FUNCTIONS

ConfigInfo::ConfigInfo()

Description: Empty constructor. Reads configuration file. returns: void

ConfigInfo::ConfigInfo(char \*atype, char \*aapphost, int asport, char \*acomport, int  
abaud\_rate, int adata\_bits, int astop\_bits, char \*aparity, char \*aserver )

Description: Constructor accepting data members as arguments. returns: void

ConfigInfo::~ConfigInfo()

Description: Destructor. deletes string members apphost, comport\_, type\_, connect\_script, server\_, and logon\_. returns: void

int ConfigInfo::write()

Description: Writes out the data members into the "com.cfg" file in the current directory. returns: void

char \*ConfigInfo::host()

Description: Accesser function. returns: host name.

char \*ConfigInfo::type()

Description: Accessor function. returns:

char \*ConfigInfo::parity()

Description: Accessor function. returns: char \* parity.

int ConfigInfo::sport()

Description: Accessor function. returns: int the socket port number.

char \*ConfigInfo::comport()

Description: Accessor function. returns: char \*, the serial comport (e.g. "com1").



char \*ConfigInfo::script()

Description: Accessor function. returns: char \*, connect script name.

char \*ConfigInfo::server()

Description: Accessor function. returns: char \*, the dui server executable name.

char \*ConfigInfo::logon()

Description: Accessor function. returns: char \*, the login name.

int ConfigInfo::baud()

Description: Accessor function. returns: int, the baud rate.

int ConfigInfo::data()

Description: Accessor function. returns: int, the data bits. (e.g. 7)

int ConfigInfo::stop()

Description: Accessor function. returns: int, the stop bits.

char \*ConfigInfo::type(char \*newtype)

Description: Data member setting function. Sets a new communications type. (i.e. "serial"). returns: new data member value.

char \*ConfigInfo::parity(char \*newparity)

Description: Data member setting function. Sets new parity value (i.e. "none"). returns: new data member value.

int ConfigInfo::sport(int newsport)

Description: Data member setting function. Sets new socket port. returns: new data member value.

char \*ConfigInfo::comport(char \*newcomport)

Description: Data member setting function. Sets new comm port (i.e. "com2"). returns: new data member value.

char \*ConfigInfo::script(char \*newscrip)

Description: Data member setting function. Sets new connect script name. returns: new data member value.

char \*ConfigInfo::server(char \*newserver)

Description: Data member setting function. Sets new server executable name. returns: new data member value.

char \*ConfigInfo::logon(char \*newlogon)

Description: Data member setting function. Sets new login name. returns: new data member value.

int ConfigInfo::baud(int newbaud)

Description: Data member setting function. Sets new baud rate. returns: new data member value.

int ConfigInfo::data(int newdata)

Description: Data member setting function. Sets new data bits (i.e. 8). returns: new data member value.

int ConfigInfo::stop(int newstop)

Description: Data member setting function. Sets new stop bits. returns: new data member value.

int ConfigInfo::inerror()

Description: Status function. returns: 0 if no error 1 if error.

## FILES

ConfigInfo.C ConfigInfo.H

### 2.1.8.6 DUI

#### NAME

DUI - Distributed User Interface

#### DESCRIPTION

DUI is a client-server system for building platform- independent user interfaces. It allows an application programmer to write a user interface that can be displayed in any environment for which a DUI client has been written. It allows the application and user interface to run on separate machines thereby distributing the user interface processing.

It was designed to separate the I/O needs of the application from the display and formatting requirements of the user interface. This greatly simplifies the task of the application programmer. It does this by providing a set of simple tools describing the basic forms of input and output an application requires to interact with a user. Some of the basic tools are:

Form - a "screen" which will contain any number of the other tools as well as a set of Command's that would operate on the data described in the form.

Command - a way for the user to act on the data entered or selected on the form.

Selection - a list of items from which the user can select one.

Multi\_Selection - a list of items from which the user can select more than one.

Toggle - a switch that can be turned either on or off.

Field - a data entry field in which the user can enter one line of data.

Text - a data entry field in which the user can enter more than one line of data.

Range - a range of values from which the user can select one.

Table - a data entry tool that allows the user to edit rows and columns in a tabular format.

These tools are used by the application and implemented by a "client". The client is a program written in the environment intended to display the interface to the user. It will implement these tools as appropriate in that environment. So even though the tools will look different in each environment for which there is a client they will still have the same functionality to the application.

## BASIC ARCHITECTURE FILES

DUI\_Command(1)  
DUI\_Command(1)

Gatec Manual

### 2.1.8.7 DUI\_Command

#### NAME

DUI\_Command - Provides support for displaying actions the user can take.

#### SYNOPSIS

```
#include "DUI_Command.H"

class DUI_Command: public DUI_Component {

communication_decls(DUI_Command)
private:
    List_of(DUI_Command) sub_commands_;
    Callback *callback_;
    DUI_View* view_;
protected:
    DUI_Command();
public:
    virtual ~DUI_Command();
    DUI_Command(const char *label, Callback *);
    DUI_Command(const char *label, DUI_Command * = 0, DUI_Command * = 0,
DUI_Command * = 0, DUI_Command * = 0, DUI_Command * = 0, DUI_Command * = 0,
DUI_Command * = 0, DUI_Command * = 0, DUI_Command * = 0, DUI_Command * = 0);
    DUI_Command(DUI_Command *, DUI_Command * = 0, DUI_Command * = 0,
DUI_Command * = 0, DUI_Command * = 0, DUI_Command * = 0, DUI_Command * = 0,
DUI_Command * = 0, DUI_Command * = 0, DUI_Command * = 0);
    DUI_Command(DUI_View *, const char *label, Callback *);
    virtual void append(DUI_Command *, DUI_Command * = 0, DUI_Command * = 0,
DUI_Command * = 0, DUI_Command * = 0, DUI_Command * = 0, DUI_Command * = 0,
DUI_Command * = 0, DUI_Command * = 0, DUI_Command * = 0);
    virtual void insert(int , DUI_Command *);
    virtual DUI_Command *remove(int );
    virtual void remove_all(int delete_commands = 1);
    virtual DUI_Command *command(int ) const;
    virtual int command_count() const;
    virtual void choose();
    virtual void execute() const;
    virtual void read_only(boolean );
    virtual void set_view(DUI_View *);
    virtual boolean children_updated() const;
    virtual void display_data(ostream & );
public:
    virtual const char *class_name() const { return "Command"; }
}
```

#### DESCRIPTION

This Class is the interface to user-accessible actions (i.e. "quit"). It

can have a call back function attached to it which is executed when the user selects it. It can contain instances of other `DUI_Commands`. In this form it is a container which serves to group other `DUI_Command`'s or `DUI_Command` groups.

## MEMBER FUNCTIONS

`DUI_Command::DUI_Command()`

Description: Empty Constructor for `DUI_Command`. returns: void

`DUI_Command::DUI_Command( const char *label, Callback *callback )`

Description: Constructor accepting command name and callback as arguments. returns: void

`Callback *cb )`

`DUI_Command::DUI_Command( DUI_View *view, const char *label, Description: This constructor is used for Commands which are imbedded in a view. returns: void`

`DUI_Command::DUI_Command( const char *label, DUI_Command *c1, DUI_Command *c2, DUI_Command *c3, DUI_Command *c4, DUI_Command *c5, DUI_Command *c6, DUI_Command *c7, DUI_Command *c8, DUI_Command *c9, DUI_Command *c10)`

Description: Constructor for creating a container command. returns: void

`DUI_Command::DUI_Command( DUI_Command *c1, DUI_Command *c2, DUI_Command *c3, DUI_Command *c4, DUI_Command *c5, DUI_Command *c6, DUI_Command *c7, DUI_Command *c8, DUI_Command *c9, DUI_Command *c10)`

Description: Constructor for creating a container command. returns: void

`DUI_Command::~~DUI_Command()`

Description: Destructor - removes sub commands if there are any. returns: void

`DUI_Command::choose()`

Description: Sets this Command as the current choice for its view. returns: void

`DUI_Command(1) Gatec Manual DUI_Command(1)`

`DUI_Command::append( DUI_Command *c1, DUI_Command *c2, DUI_Command *c3, DUI_Command *c4, DUI_Command *c5, DUI_Command *c6, DUI_Command *c7, DUI_Command *c8, DUI_Command *c9, DUI_Command *c10 )`

Description: Appends up to 10 commands to `sub_commands_` list. returns: void

`DUI_Command::insert(int i, DUI_Command *command )`

Description: Inserts a subcommand at `i` in `sub_commands_`. returns: void

`DUI_Command::remove( int i )`

Description: Removes subcommand `i`. returns: void

`DUI_Command::remove_all( int delete_commands )`

Description: Removes all subcommands, deleting them if `delete_commands` is non-zero. returns: void

DUI\_Command::command( int i )

Description: Returns DUI\_Command pointer indexed by i in sub\_commands\_; returns: sub-command indexed by i or 0 if out of range.

DUI\_Command::command\_count()

Description: Gives the number of DUI\_Commands in the subcommand list. returns: the number of subcommands;

DUI\_Command::execute()

Description: Executes this commands call back function if any. returns: void

DUI\_Command::read\_only( boolean ro )

Description: Sets this command and all of its sub commands to arg ro if not already set. returns: void

DUI\_Command::set\_view( DUI\_View \*view )

Description: Sets the view of this command and all its subcommands to arg view. returns: void

DUI\_Command::children\_updated()

Description: Indicate whether this command has been updated. returns: 1 if this or any of it's children is updated()

DUI\_Command::display\_data( ostream &out )

Description: Outputs the command and subcommands to arg out in a simple textual format. returns: void

## FILES

DUI\_Command.C DUI\_Command.H

## 2.1.8.8 DUI\_Component

### NAME

DUI\_Component - Base class for DUI\_Widgets that can be attached to a DUI\_View.

### SYNOPSIS

```
#include "DUI_Component.H"

class DUI_Component: public DUI_Widget {

communication_decls(DUI_Component)
private:
    boolean    read_only_;
    Callback   *update_callback;
public:
    virtual ~DUI_Component();
    virtual void read_only(boolean );
    virtual boolean read_only() const;
    virtual boolean children_updated() const { return updated(); };
    virtual const char *check_invalid();
    virtual void active_update(Callback *);
protected:
    DUI_Component();
    DUI_Component(const char *name);
protected:
    friend class Session;
    virtual void receive();
public:
    virtual const char *class_name() const { return "Component"; }
}
```

### DESCRIPTION

All the DUI\_Widgets that can be attached to a DUI\_View are subclassed off of this class. It eliminates the need for DUI\_View and DUI\_Group to know what kind of components they are dealing with for certain operations. It is derived from DUI\_Widget(which see). See also DUI.

### MEMBER FUNCTIONS

DUI\_Component::DUI\_Component()

Description: Constructor for DUI\_Component which is a DUI\_Widget. returns: void

DUI\_Component::DUI\_Component( const char \*name )

Description: Constructor which accepts a name passing it on to DUI\_Widget which actually stores the name. returns: void

DUI\_Component::~~DUI\_Component()

Description: Destructor. returns: void

DUI\_Component::read\_only( boolean ro )

Description: Sets the read only flag to arg. This is where read only-ness is stored. returns: void

DUI\_Component::read\_only()

Description: Retrieve read only status. returns:  
boolean representing the read only status.

DUI\_Component::check\_invalid()

Description: check\_invalid() should return 0 if a components value is valid, otherwise it should return a const char \* description of why it's invalid. This is a virtual class to be defined appropriately by derived classes. It has no meaning for DUI\_Component. returns:  
0 always.

DUI\_Component::receive()

Description: Default receive() function calls update\_callback if it is not 0. The receive function is called whenever this component is received either by the client or application. returns: void

DUI\_Component::active\_update( Callback \*callback )

Description: This provides a way to set the active update Callback. The active update callback is a function called whenever this object is received by the application. (See Callback(1)). returns:  
void

## FILES

DUI\_Component.C DUI\_Component.H



### 2.1.8.9 DUI\_Dialog

#### NAME

DUI\_Dialog - Specialized DUI\_View for informational and confirmation dialogs.

#### SYNOPSIS

```
#include "DUI_Dialog.H"

class DUI_Dialog: public DUI_View {

communication_decls(DUI_Dialog)
public:
    static DUI_Dialog *instance( const char *label, DUI_Component *comp = 0, const char
*command1 = 0, Callback *callback1 = 0, const char *command2 = 0, Callback *callback2 = 0
);
    static DUI_Dialog *instance( const char *label, Callback *yes_callback, Callback *
no_callback = 0 );
    void add_command(const char *command, Callback *callback = 0);
    virtual ~DUI_Dialog();
private:
    static DUI_Dialog *instance_;
    DUI_Label *label_component;
    DUI_Dialog();
    DUI_Dialog(char *);
    void change_dialog(const char *label, DUI_Component *comp, const char *command1 =
0, Callback *callback1 = 0, const char *command2 = 0, Callback *callback2 = 0);
public:
    virtual const char *class_name() const { return "Dialog";
}
}
```

#### DESCRIPTION

This class is intended to provide easy access to a DUI\_View(which see) for displaying informational messages and simple confirmations. It contains only two view level commands. One of its constructors allows for the addition of view level component so it can be used to construct a more complex view.

#### MEMBER FUNCTIONS

DUI\_Dialog \*DUI\_Dialog::instance( const char \*label, DUI\_Component \*comp, const char \*command1, Callback \*callback1, const char \*command2, Callback \*callback2 )

Description: There is only one instance of a Dialog in any application. The same dialog is used each time but is modified according to the arguments of the instance function. This function allows the user to tailor the dialog's component as well as the two

view level commands. All but the first argument default to 0 so it can also be used to display a message with a single OK button. (see `change_dialog`). returns: void

`DUI_Dialog::instance( const char *label, Callback *yes_callback, Callback * no_callback )` Description: This instance function creates a `DUI_Dialog` with "Yes" and "No" buttons that use the callbacks passed in as arguments. returns: void

`DUI_Dialog::change_dialog( const char *label, DUI_Component *comp, const char *command1, Callback *callback1, const char *command2, Callback *callback2 )` Description: This function changes the contents of the dialog. If the `DUI_Component*` argument is 0 it sets the dialogs component to the label argument otherwise it uses the component passed in and sets the dialog name to label. Also if `command1` is 0 it adds a default command named "OK" to the dialog. returns: void

`DUI_Dialog::DUI_Dialog()`  
Description: Empty constructor. returns: void

`DUI_Dialog::DUI_Dialog( char *view_name)`  
Description: Constructor accepting a name as an argument. returns: void

`void DUI_Dialog::add_command(const char *cmd, Callback *callback)`  
Description: Adds a new `End_Command`(which see) to this dialog's command group. returns: void

`DUI_Dialog::~~DUI_Dialog()`  
Description: Destructor. Resets `instance_` pointer to 0 and this view's component to 0 so the component won't be deleted by the `DUI_View` destructor. returns: void

## FILES

`DUI_Dialog.C` `DUI_Dialog.H`

### 2.1.8.10 DUI\_End\_Command

#### NAME

DUI\_End\_Command - A command which causes the view to close down upon selection.

#### SYNOPSIS

```
#include "DUI_End_Command.H"
```

```
class DUI_End_Command: public DUI_Command {
```

```
communication_decls(DUI_End_Command)
```

```
private:
```

```
    DUI_End_Command();
```

```
public:
```

```
    virtual ~DUI_End_Command();
```

```
    DUI_End_Command(const char *label, Callback *);
```

```
    DUI_End_Command(const char *label, DUI_Command * = 0, DUI_Command * = 0,  
DUI_Command * = 0, DUI_Command * = 0, DUI_Command * = 0, DUI_Command * = 0,  
DUI_Command * = 0, DUI_Command * = 0, DUI_Command * = 0, DUI_Command * = 0);
```

```
    DUI_End_Command(DUI_Command *, DUI_Command * = 0, DUI_Command * = 0,  
DUI_Command * = 0, DUI_Command * = 0, DUI_Command * = 0, DUI_Command * = 0,  
DUI_Command * = 0, DUI_Command * = 0, DUI_Command * = 0);
```

```
    DUI_End_Command(DUI_View *, const char *label, Callback *);
```

```
public:
```

```
    virtual const char *class_name() const { return "End_Command"; }
```

```
}
```

#### DESCRIPTION

This class is derived from DUI\_Command (which see) that causes the view to be closed down on the client side when it is selected.

#### MEMBER FUNCTIONS

DUI\_End\_Command::DUI\_End\_Command()

Description: Constructor accepting a name and a Call- back function as args. returns: void

\*callback )

DUI\_End\_Command::DUI\_End\_Command( const char \*label, Callback Description: Constructor accepting a name and a Call- back function as args. returns: void

Callback \*cb )

DUI\_End\_Command::DUI\_End\_Command( DUI\_View \*v, const char \*l, Description: This constructor is used for Commands which are imbedded in a view component. returns: void

\*c1, DUI\_End\_Command::DUI\_End\_Command( const char \*label, DUI\_Command  
DUI\_Command \*c2, DUI\_Command \*c3, DUI\_Command \*c4, DUI\_Command

\*c5, DUI\_Command \*c6, DUI\_Command \*c7, DUI\_Command \*c8, DUI\_Command \*c9, DUI\_Command \*c10)

Description: Constructor accepting a name and upto ten DUI\_Commands as args. It creates a container for other DUI\_Commands. returns: void

DUI\_End\_Command::DUI\_End\_Command( DUI\_Command \*c1, DUI\_Command \*c2, DUI\_Command \*c3, DUI\_Command \*c4, DUI\_Command \*c5, DUI\_Command \*c6, DUI\_Command \*c7, DUI\_Command \*c8, DUI\_Command \*c9, DUI\_Command \*c10)

Description: Constructor which accepts ten commands as args without a name. returns: void

DUI\_End\_Command::~~DUI\_End\_Command()

Description: Destructor which at present does nothing. returns: void

## FILES

DUI\_End\_Command.C DUI\_End\_Command.H

### 2.1.8.11 DUI\_Field

#### NAME

DUI\_Field - Provides support for a data entry field.

#### SYNOPSIS

```
#include "DUI_Field.H"

class DUI_Field: public DUI_Component {

communication_decls(DUI_Field)
private:
    STRING *validation_;
    int   max_length_;
    boolean mandatory_;
    List_of(Constraint) constraints;
    List_of(Modifier) modifiers;
protected:
    STRING*   value_;
    DUI_Field();
public:
    DUI_Field(const char *name, const char *sample_value = 0, int max_length = 0);
    DUI_Field(const char *name, int max_length);
    virtual ~DUI_Field();
    virtual void value(const char *new_value);
    virtual void clear_value();
    virtual const char *value() const;
    virtual const char *invalid();
    virtual const char *check_invalid();
    virtual void max_length(int length);
    virtual int max_length() const;
    virtual void mandatory(boolean );
    virtual boolean mandatory() const;
    virtual void is(const Modifier *);
    virtual void is(const Constraint *);
    virtual void display_data(ostream & );
public:
    virtual const char *class_name() const { return "Field";
    }
}
```

#### DESCRIPTION

This class is used when the application needs textual input from the user that will no exceed one line. It can have any number of Modifiers and Constraints(which see) attached to it that either modify the input after the user has entered it or does not allow the user to enter invalid input. Example constraints are Integer, and Date(which see). If the programmer needs multi-line input see

DUI\_Text.

## MEMBER FUNCTIONS

DUI\_Field::DUI\_Field()

Description: Empty constructor. returns: void

int maxlen )

DUI\_Field::DUI\_Field( const char \*name, const char \*sample\_value, Description: Constructor accepting a name, initial value and maximum length as arguments. returns: void

DUI\_Field::DUI\_Field( const char \*name, int maxlen )

Description: Constructor accepting a name and maximum length as arguments. returns: void

DUI\_Field::~~DUI\_Field()

Description: Destructor which value string error string and all constraints and modifiers. returns: void

const char \*DUI\_Field::check\_invalid()

Description: Checks all constraints to make sure this field value is valid. if it is not it sets an error string and returns it. returns: error string or 0.

const char \*DUI\_Field::invalid()

Description: returns 0 if the previous new\_value passed to value() was valid otherwise returns an explanation of why new\_value is invalid. returns: 0 or error string.

const char \*DUI\_Field::value()

Description: returns the value in the field -- ALWAYS returns a valid value. (initial empty field IS valid)  
returns: void

void DUI\_Field::max\_length( int maxlen )

Description: Sets maximum length for this field. returns: void

DUI\_Field::mandatory( boolean man )

Description: Sets mandatory flag for this field. returns: void

DUI\_Field::mandatory()

Description: Mandatory flag accessor function. returns: boolean  
Mandatory flag.

int DUI\_Field::max\_length()

Description: Returns the maximum length of the value in this field  
returns: int maximum length.

void DUI\_Field::is( const Constraint \*constraint )

Description: allows the application to specify a Constraint that the field data MUST conform to. (i.e. Regular\_Expression("[0-9]\*") ) returns: void

void DUI\_Field::is( const Modifier \*modifier )

Description: Allows the application to specify a Modifier for the field data (i.e. Lower\_Case, Truncated, Left\_Justified) returns: void

void DUI\_Field::value( const char \*new\_value )

Description: Assign new\_value to this DUI\_Field. - modifies new\_value with all Modifiers then tests new\_value against each Constraint. If new\_value conforms to all Constraints, it is saved as the new value. validate() may be called to test success of failure of this call returns: void

DUI\_Field::clear\_value()

Description: Clears the value of this field, does NOT check modifiers and constraints. returns: void

DUI\_Field::display\_data( ostream &out )

Description: Prints name and value to stream. (e.g. "FieldName: some value0"). returns: void

## FILES

DUI\_Field.C DUI\_Field.H

### 2.1.8.12 DUI\_Form

#### NAME

DUI\_Form - Entry class for application created views.

#### SYNOPSIS

```
#include "DUI_Form.H"

class DUI_Form: public DUI_View {

communication_decls(DUI_Form)
protected:
    DUI_Form();
public:
    virtual ~DUI_Form();
    DUI_Form(const char *label, DUI_Component *component = 0, DUI_Command *cmd = 0);
public:
    virtual const char *class_name() const { return "Form"; }
}
```

#### DESCRIPTION

This is the class the application would derive its views from. It is derived from DUI\_View(which see).

#### MEMBER FUNCTIONS

DUI\_Form::DUI\_Form()

Description: Empty Constructor for DUI\_Form. returns:  
void

DUI\_Form::DUI\_Form( const char \*label,  
DUI\_Component \*component, DUI\_Command \*command)

Description: Constructor accepting a name, a component (the body of the view), and a command (the view level commands.).  
returns:  
void

DUI\_Form::~~DUI\_Form()

Description: Destructor. Does nothing. returns: void

#### FILES

DUI\_Form.C DUI\_Form.H



### 2.1.8.13 DUI\_Group

#### NAME

DUI\_Group - A grouping class for DUI\_Components.

#### SYNOPSIS

```
#include "DUI_Group.H"

class DUI_Group: public DUI_Component {

communication_decls(DUI_Group)
private:
    List_of(DUI_Component) components;
    STRING *validation_;
public:
    virtual ~DUI_Group();
    DUI_Group(const char *name, DUI_Component *c1 = 0, DUI_Component *c2 = 0,
DUI_Component *c3 = 0, DUI_Component *c4 = 0, DUI_Component *c5 = 0,
DUI_Component *c6 = 0, DUI_Component *c7 = 0, DUI_Component *c8 = 0,
DUI_Component *c9 = 0, DUI_Component *c10 = 0);
    DUI_Group(DUI_Component *c1, DUI_Component *c2 = 0, DUI_Component *c3 = 0,
DUI_Component *c4 = 0, DUI_Component *c5 = 0, DUI_Component *c6 = 0,
DUI_Component *c7 = 0, DUI_Component *c8 = 0, DUI_Component *c9 = 0,
DUI_Component *c10 = 0);
    virtual void append(DUI_Component *c1, DUI_Component *c2 = 0, DUI_Component *c3
= 0, DUI_Component *c4 = 0, DUI_Component *c5 = 0, DUI_Component *c6 = 0,
DUI_Component *c7 = 0, DUI_Component *c8 = 0, DUI_Component *c9 = 0,
DUI_Component *c10 = 0);
    virtual void append(const char *label);
    virtual void insert(int i, DUI_Component *c);
    virtual void insert(int i, const char *label);
    virtual DUI_Component *remove(int i);
    virtual void remove_all(int delete_components = 1);
    virtual DUI_Component *component(int i) const;
    virtual int component_count() const;
    virtual const char *check_invalid();
    virtual void read_only(boolean );
    virtual boolean children_updated() const;
    virtual int component_index(const DUI_Component *) const;
    virtual int component_index(const char *) const;
    virtual void display_data(ostream & );
protected:
    DUI_Group();
public:
    virtual const char *class_name() const { return "Group";
}
}
```

#### DESCRIPTION

This class is used to group `DUI_Components` together. It is derived from `DUI_Components` so it may contain instances of other `DUI_Groups`. Since `DUI_Views`(which see) contain only one top level component, it is usually a `DUI_Group` containing the rest of the components which make up the body of the view.

## MEMBER FUNCTIONS

`DUI_Group::DUI_Group()`

Description: Empty Constructor. returns: void

`DUI_Group::DUI_Group( DUI_Component *c1, DUI_Component *c2, DUI_Component *c3, DUI_Component *c4, DUI_Component *c5, DUI_Component *c6, DUI_Component *c7, DUI_Component *c8, DUI_Component *c9, DUI_Component *c10)`

Description: Constructor accepting upto ten components. returns: void

`DUI_Group::DUI_Group( const char *name, DUI_Component *c1, DUI_Component *c2, DUI_Component *c3, DUI_Component *c4, DUI_Component *c5, DUI_Component *c6, DUI_Component *c7, DUI_Component *c8, DUI_Component *c9, DUI_Component *c10)`

Description: Constructor accepting upto ten components and a name. returns: void

`DUI_Group::~DUI_Group()`

Description: Destructor. Removes all components. returns: void

`DUI_Group::append( DUI_Component *c1, DUI_Component *c2, DUI_Component *c3, DUI_Component *c4, DUI_Component *c5, DUI_Component *c6, DUI_Component *c7, DUI_Component *c8, DUI_Component *c9, DUI_Component *c10)`

Description: Appends upto ten components to the group. returns: void

`DUI_Group::append( const char *label )`

Description: Function that appends a `DUI_Label` (which see) to the group given a character string. returns: void

`DUI_Group::insert( int i, DUI_Component *c )`

Description: Allows the insertion of a `DUI_Component` at a returns

FILES

`DUI_Group.C` `DUI_Group.H`

### 2.1.8.14 DUI\_Invisible\_Field

#### NAME

DUI\_Invisible\_Field - A DUI\_Field that does not display when edited.

#### SYNOPSIS

```
#include "DUI_Invisible_Field.H"

class DUI_Invisible_Field: public DUI_Field {

communication_decls(DUI_Invisible_Field)
protected:
    DUI_Invisible_Field();
    const char *scramble(const STRING *) const;
public:
    DUI_Invisible_Field(const char *name, const char *sample_value = 0);
    virtual ~DUI_Invisible_Field();
    virtual void value(const char *new_value);
    virtual const char *value() const;
    virtual void is(const Modifier *);
    virtual void display_data(ostream & );
public:
    virtual const char *class_name() const { return "Invisible_Field"; }
}
```

#### DESCRIPTION

This class is derived from DUI\_Field(which see). It provides support for fields that contain sensitive data. Its contents are communicated to client in a scrambled form. Its contents are not visible on the clients side.

#### MEMBER FUNCTIONS

DUI\_Invisible\_Field::DUI\_Invisible\_Field()  
Description: Empty constructor. returns: void

DUI\_Invisible\_Field::DUI\_Invisible\_Field( const char \*name, const char  
\*sample\_value )  
Description: Constuctor accepting a name and an initial value. returns: void

DUI\_Invisible\_Field::~~DUI\_Invisible\_Field()  
Description: Destructor. Does nothing. returns: void

DUI\_Invisible\_Field::value( const char \*new\_value )  
Description: Assignment of new value, recomputes value\_. calls DUI\_Field::value().  
returns: void

DUI\_Invisible\_Field::value()

Description: returns value (unscrambled) returns: char \* value.

DUI\_Invisible\_Field::is( const Modifier \* modifier )

Description: Adds a modifier to the field. returns:  
void

DUI\_Invisible\_Field::scramble( const STRING \*unscrambled )

Description: returns scrambled version of unscrambled string, uses STRING::buf() returns:  
char \* scrambled value.

DUI\_Invisible\_Field::display\_data( ostream &out )

Description: Displays only field name. Contents is sensitive. returns: void

## FILES

DUI\_Invisible\_Field.C DUI\_Invisible\_Field.H

DUI(1)      Last change: Tue Jan 4 16:20:30 1994      2

### 2.1.8.15 DUI\_Label

#### NAME

DUI\_Label - Displays non-editable text.

#### SYNOPSIS

```
#include "DUI_Label.H"

class DUI_Label: public DUI_Component {

communication_decls(DUI_Label)
protected:
    DUI_Label();
public:
    virtual ~DUI_Label();
    DUI_Label(const char *name);
public:
    virtual const char *class_name() const { return "Label";
}
}
```

#### DESCRIPTION

This class is used for displaying text on the view that the user will not edit.

#### MEMBER FUNCTIONS

DUI\_Label::DUI\_Label()

Description: Empty Constructor. returns: void

DUI\_Label::DUI\_Label(const char \*name)

Description: Constructor accepting the text to be displayed.  
returns: void

DUI\_Label::~~DUI\_Label()

Description: Destructor. Does nothing. returns: void

#### FILES

DUI\_Label.C DUI\_Label.H

### 2.1.8.16 DUI\_Multi\_Selection

#### NAME

DUI\_Multi\_Selection - A list accepting multiple selections.

#### SYNOPSIS

```
#include "DUI_Multi_Selection.H"

class DUI_Multi_Selection: public DUI_Selection {

communication_decls(DUI_Multi_Selection)
private:
    DUI_Group *selected_group;
public:
    virtual ~DUI_Multi_Selection();
    DUI_Multi_Selection(const char *name, const char * = 0, const char * = 0, const char * = 0,
const char * = 0, const char * = 0, const char * = 0, const char * = 0, const char * = 0, const char
* = 0, const char * = 0);
    DUI_Multi_Selection(const char *name, DUI_Component *, DUI_Component * = 0,
DUI_Component * = 0, DUI_Component * = 0, DUI_Component * = 0, DUI_Component * = 0,
DUI_Component * = 0, DUI_Component * = 0, DUI_Component * = 0, DUI_Component * =
0);
    DUI_Multi_Selection(const char *name, DUI_Group *group);
    virtual void select(const DUI_Component *);
    virtual void select_all();
    virtual void deselect();
    virtual DUI_Component *remove(int );
    virtual void deselect(const DUI_Component *);
    virtual void deselect(const char *);
    virtual DUI_Group *selections();
    virtual int selection_index(const DUI_Component *);
    virtual void display_data(ostream & );
protected:
    DUI_Multi_Selection();
public:
    virtual const char *class_name() const { return "Multi_Selection"; }
}
```

#### DESCRIPTION

This class is used when the applications has a list of things from which the user can choose more than one. It is derived from DUI\_Selection(which see) which allows the user to choose only one.

#### MEMBER FUNCTIONS

DUI\_Multi\_Selection::DUI\_Multi\_Selection()  
Description: Empty constructor. returns: void

DUI(1) Last change: Tue Jan 4 16:20:26 1994 1

DUI\_Multi\_Selection(1) Gatec Manual DUI\_Multi\_Selection(1)

DUI\_Multi\_Selection::DUI\_Multi\_Selection( const char \*name, DUI\_Component \*c1, DUI\_Component \*c2, DUI\_Component \*c3, DUI\_Component \*c4, DUI\_Component \*c5, DUI\_Component \*c6, DUI\_Component \*c7, DUI\_Component \*c8, DUI\_Component \*c9, DUI\_Component \*c10) Description: Constructor accepting a name, and ten Components as arguments. returns: void

char \*c1, DUI\_Multi\_Selection::DUI\_Multi\_Selection( const char \*name, const char \*c2, const char \*c3, const char \*c4, const char \*c5, const char \*c6, const char \*c7, const char \*c8, const char \*c9, const char \*c10) Description: Constructor accepting a name and ten strings as arguments. returns: void

DUI\_Group \*new\_group )

DUI\_Multi\_Selection::DUI\_Multi\_Selection( const char \*name, Description: Constructor accepting a name and a DUI\_Group representing its selections as arguments. returns: void

DUI\_Multi\_Selection::~~DUI\_Multi\_Selection()

Description: Destructor. Does nothing. returns: void

DUI\_Multi\_Selection::remove( int i )

Description: removes ith component from select and returns pointer to it if ith component was selected, deselect() is called to avoid dangling ptr. returns: DUI\_Component indexed by i or 0.

DUI\_Multi\_Selection::select( const DUI\_Component \*component )

Description: Selects a DUI\_Component (if not already selected). returns: void

DUI\_Multi\_Selection::select\_all()

Description: select all - turn off updates until all unselected elements have been appended to selected\_group. returns: void

DUI\_Multi\_Selection::deselect()

Description: Deselect all. returns: void

DUI\_Multi\_Selection::deselect( const DUI\_Component \* component )

Description: Deselect a DUI\_Component returns: void

DUI\_Multi\_Selection::deselect( const char \* label )

Description: Deselect a DUI\_Component by name (provided for application programmer) returns: void

DUI\_Multi\_Selection::selections()

Description: Returns group of selected components. returns: DUI\_Group \* selected components.

ponent )

DUI\_Multi\_Selection::selection\_index( const DUI\_Component \* com- Description: returns:

order that component was selected in, -1 if not selected

DUI\_Multi\_Selection::display\_data( ostream &out )

Description: prints selection in simple ascii format to stream. returns: void

## FILES

DUI\_Multi\_Selection.C DUI\_Multi\_Selection.H



### 2.1.8.17 DUI\_Range

#### NAME

DUI\_Range - allows the user to select from a range of values.

#### SYNOPSIS

```
#include "DUI_Range.H"

class DUI_Range: public DUI_Component {

communication_decls(DUI_Range)
private:
    int  lower_;
    int  upper_;
    int  value_;
protected:
    DUI_Range();
public:
    virtual ~DUI_Range();
    DUI_Range(const char *name, int lower = 0, int upper = 10);
    int lower() const;
    int upper() const;
    int value() const;
    void lower(int );
    void upper(int );
    void value(int );
    virtual void display_data(ostream & );
public:
    virtual const char *class_name() const { return "Range";
    }
}
```

#### DESCRIPTION

This class is used when the application wants to ask the user for a value within a particular range. The value and upper and lower limits are expressed as integers.

#### MEMBER FUNCTIONS

DUI\_Range::DUI\_Range()

Description: Empty constructor. returns: void

DUI\_Range::DUI\_Range( const char \*name, int l, int u )

Description: Constructor accepting a name and an upper and lower limit. returns: void

DUI\_Range::~DUI\_Range()

Description: Destructor. Does nothing. returns: void

int DUI\_Range::lower()

Description: Member access function. returns: int lower limit.

Description:

Member access function. returns: int upper limit.

Description:

Member access function. returns: int current value.

void DUI\_Range::lower(int l)

Description: Member setting function for lower limit. returns:  
void

Description:

Member setting function for upper limit. returns: void

Description:

Member setting function for current value. returns:  
void

DUI\_Range::display\_data( ostream &out )

Description: Prints a simple ASCII representation of this range  
and its current value on stream "out". returns: void

## FILES

DUI\_Range.C DUI\_Range.H



}

## DESCRIPTION

This class is used when the application needs to display a list of items from which the user can select one. It is a base class for `DUI_Multi_Selection` (which see) which allows the user to select more than one of the listed items.

## MEMBER FUNCTIONS

`DUI_Selection::DUI_Selection()`

Description: Empty Constructor. returns: void

`DUI_Selection::DUI_Selection( const char *name, DUI_Component *c1, DUI_Component *c2, DUI_Component *c3, DUI_Component *c4, DUI_Component *c5, DUI_Component *c6, DUI_Component *c7, DUI_Component *c8, DUI_Component *c9, DUI_Component *c10)` Description:

Constructor accepting a name and ten components as items in its list. returns: void

`DUI_Selection::DUI_Selection( const char *name, const char *c1, const char *c2, const char *c3, const char *c4, const char *c5, const char *c6, const char *c7, const char *c8, const char *c9, const char *c10)` Description: Constructor accepting a name and ten strings as items in its list. returns: void

`*new_group )`

`DUI_Selection::DUI_Selection( const char *name, DUI_Group` Description: Constructor accepting a name and a group to use as the item list. returns: void

`DUI_Selection::~~DUI_Selection()`

Description: Destructor which deletes its associated list. returns: void

`DUI_Selection::append( DUI_Component *c1, DUI_Component *c2, DUI_Component *c3, DUI_Component *c4, DUI_Component *c5, DUI_Component *c6, DUI_Component *c7, DUI_Component *c8, DUI_Component *c9, DUI_Component *c10 )` Description: Add upto ten `DUI_Components` to the item list. returns: void

`DUI_Selection::insert( int i, DUI_Component * component )`

Description: Insert component into list at index i. returns: void

`void DUI_Selection::append( const char * label )`

Description: Append string "label" to item list. returns: void

`void DUI_Selection::insert( int i, const char * label )`

Description: Insert string "label" into item list at index i. returns: void

`DUI_Selection::remove( int i )`

Description: Removes ith component from select and returns pointer to it. If ith component was selected, `deselect()` is called to avoid dangling ptr returns:

`DUI_Component *` the component removed.

DUI\_Selection::remove\_all( int delete\_components )

Description: Removes all components from selection. WARNING: Does NOT delete components returns: void

void DUI\_Selection::deselect()

Description: Resets selection so none are selected. returns: void

void DUI\_Selection::select( const DUI\_Component \*component )

Description: Selects a component given a pointer to it, if it is not selected. Does not check to see if component is in its list. returns: void

void DUI\_Selection::select( const char \* label )

Description: Select a DUI\_Component by name (provided for application programmer). returns: void

DUI\_Selection::selection()

Description: Accesser function. returns: the selected component.

DUI\_Selection::component( int i )

Description: Accesser function. returns: component indexed by i in the item list.

DUI\_Selection::component\_count()

Description: Accessor function. returns: The number of items in the list.

DUI\_Selection::set\_group( DUI\_Group \*new\_group )

Description: Changes selection to use new\_group as its list. WARNING: This function does NOT delete the previous group returns: void

DUI\_Selection::component\_index( DUI\_Component \*component )

Description: Accesser function. returns: the index of a component by pointer or -1 if component not found.

DUI\_Selection::component\_index( const char \*label )

Description: Accesser function. returns: the index of a component by name or -1 if component not found.

DUI\_Selection::children\_updated()

Description: Status function. returns: 1 if this or any of it's children is updated()

DUI\_Selection::display\_data( ostream &out )

Description: Prints the list of items and the one selected onto the stream "out". returns: void

## FILES

DUI\_Selection.C DUI\_Selection.H

### 2.1.8.19 DUI\_Table

#### NAME

DUI\_Table - For displaying and editing a table of data with rows and columns.

#### SYNOPSIS

```
#include "DUI_Table.H"
```

```
class DUI_Table: public DUI_Component {
```

```
communication_decls(DUI_Table)
```

```
private:
```

```
    STRING *validation_;
```

```
    List_of(Table_Column) columns;
```

```
protected:
```

```
    DUI_Table();
```

```
public:
```

```
    DUI_Table(const char *name, int num_rows = 1, int num_columns = 1);
```

```
    DUI_Table(const char *name, int num_rows, const char *col1, const char *col2 = 0,  
const char *col3 = 0, const char *col4 = 0, const char *col5 = 0, const char *col6 = 0, const char  
*col7 = 0, const char *col8 = 0, const char *col9 = 0, const char *col10 = 0);
```

```
    virtual ~DUI_Table();
```

```
    virtual int row_count() const;
```

```
    virtual void reset_row_count(int num_rows);
```

```
    virtual void append_row(List_of(STRING) *values = 0);
```

```
    virtual void insert_row(int row, List_of(STRING) *values = 0);
```

```
    virtual void remove_row(int row);
```

```
    virtual int column_count() const;
```

```
    virtual void append_column(const char *name = "");
```

```
    virtual void insert_column(int col, const char *name = "");
```

```
    virtual void remove_column(int col);
```

```
    virtual void column_name(int col, const char *);
```

```
    virtual const char *column_name(int col);
```

```
    virtual void column_is(int col, Modifier *);
```

```
    virtual void column_is(int col, Constraint *);
```

```
    virtual void value(int row, int col, const char *);
```

```
    virtual const char *value(int row, int col) const;
```

```
    virtual void clear_column_values(int col);
```

```
    virtual void clear_row_values(int row);
```

```
    virtual void clear_values();
```

```
    virtual const char *invalid();
```

```
    virtual const char *check_invalid();
```

```
    virtual void add_row_ok(boolean );
```

```
    virtual void remove_row_ok(boolean );
```

```
    virtual void change_ok(boolean );
```

```
    virtual void change_column_ok(int column, boolean );
```

```
    virtual boolean add_row_ok() const;
```

```

virtual boolean remove_row_ok() const;
virtual boolean change_ok() const;
virtual boolean change_column_ok(int column) const;
virtual boolean read_only() const;
virtual void read_only(boolean );
virtual void display_data(ostream & );
public:
    virtual const char *class_name() const { return "Table";
}
}

```

## DESCRIPTION

This class is used if the application programmer wants to display data to or accept data from the user in the form of a table with many rows having the same number and type of columns. It allows for the editing of each column in any of the rows, as well as deleting and inserting rows. Any of the operations on the table can be enabled or disabled. All columns can have Constraints(which see) and Modifiers(which see) attached to them. (See also Table\_Column).

## MEMBER FUNCTIONS

DUI\_Table::DUI\_Table()

Description: Empty Constructor. returns: void

num\_columns )

DUI\_Table::DUI\_Table( const char \*name, int num\_rows, int Description: Constructor accepting a name, the number of rows, and the number columns. returns: void

\*col1, DUI\_Table::DUI\_Table( const char \* name, int num\_rows, const char const char \*col2, const char \*col3, const char \*col4, const char \*col5, const char \*col6, const char \*col7, const char \*col8, const char \*col9, const char \*col10) Description: Constructor accepting table name, and column names as arguments. returns: void

DUI\_Table::~~DUI\_Table()

Description: Destructor for DUI\_Table. Deletes validation string. returns: void

DUI\_Table::row\_count()

Description: Accesser function. returns: number of rows in table.

DUI\_Table::reset\_row\_count( int num\_rows )

Description: Sets the number of rows in the table. Dropping the remaining rows. returns: void

void DUI\_Table::append\_row(List\_of(String))

Description: Appends a row with values. Accepts a pointer to a list of String's. (which see). returns: void

DUI\_Table::insert\_row( int row, List\_of(String))

Description: Inserts a row with values into the table at index "row". returns: void

DUI\_Table::remove\_row( int row )

Description: Removes the row indexed by "row" from table. returns: void

DUI\_Table::column\_count()

Description: Accessor function. returns: the number of columns.

DUI\_Table::append\_column( const char \*name )

Description: Appends a column with 'name' to table. returns: void

DUI\_Table::insert\_column( int col, const char \*name )

Description: Inserts a column with "name" into table. returns: void

DUI\_Table::remove\_column( int col )

Description: Removes the column indexed by "col" from table. returns: void

DUI\_Table::column\_name( int col, const char \*name )

Description: Sets the name of a column. returns: void

DUI\_Table::column\_name( int col )

Description: Accesser function. returns: the name of the column indexed by "col".

DUI\_Table::column\_is(int col, Constraint \* constraint)

Description: Adds a Constraint to a column. Columns can have constraints and modifiers like DUI\_Fields(which see). returns: void

DUI\_Table::column\_is(int col, Modifier \* modifier )

Description: Adds a Modifier to the column indexed by "col". returns: void

DUI\_Table::value( int row, int col, const char \*new\_value )

Description: Sets the value at "row", "col". returns:  
void

DUI\_Table::value( int row, int col )

Description: Accesser function. returns: value at "row", "col" if not there 0.

DUI\_Table::clear\_column\_values( int col )

Description: Clears values for the column indexed by "col". returns: void

DUI\_Table::clear\_row\_values( int row )

Description: Clear values for the row indexed by "row". returns: void

DUI\_Table::clear\_values()

Description: Clears all values in the table. returns:  
void

DUI\_Table::invalid()

Description: Status function. returns: 0 if the previous new\_value passed to value() was valid otherwise returns an explanation of why new\_value is invalid



DUI\_Table::check\_invalid()

Description: Ensures values in DUI\_Table are valid returns: void

DUI\_Table::add\_row\_ok( boolean b )

Description: Sets permissions on adding rows. returns:  
void

DUI\_Table::remove\_row\_ok( boolean b )

Description: Sets permissions on deleting rows. returns: void

DUI\_Table::change\_ok( boolean b )

Description: Sets permission on whether any of the rows can be changed. returns: void

DUI\_Table::add\_row\_ok()

Description: Accesser function. returns: adding row permission.

DUI\_Table::remove\_row\_ok()

Description: Accesser function. returns: delete row permissions.

DUI\_Table::change\_ok()

Description: Accesser function. returns: change permissions.

DUI\_Table::read\_only()

Description: Accesser function. returns: read only status.

DUI\_Table::read\_only( boolean b )

Description: Sets read only status to 0 if arg is true else it sets OK on all the other permissions.  
returns:  
void

DUI\_Table::change\_column\_ok( int column, boolean ok )

Description: Sets change for the column indexed by "column". Each column has its own change permission flag. returns: void

DUI\_Table::change\_column\_ok( int column )

Description: Accesser function. returns: permission for column "column".

DUI\_Table::display\_data( ostream &out )

Description: Prints a simple ascii representation of the table onto stream "out". returns: void

## FILES

DUI\_Table.C DUI\_Table.H

### 2.1.8.20 DUI\_Text

NAME

DUI\_Text - Provides support for multi-line text editing.

SYNOPSIS

```
#include "DUI_Text.H"

class DUI_Text: public DUI_Component {

communication_decls(DUI_Text)
private:
    List_of(STRING) lines_;
    List_of(Modifier) modifiers;
protected:
    DUI_Text();
public:
    DUI_Text(const char *name, const char *sample_value = 0);
    virtual ~DUI_Text();
    virtual void lines(int i, const char *new_lines);
    virtual void line(int i, const char *new_line);
    virtual void remove_lines(int first, int last = 0);
    virtual void insert_line(int i, const char *new_line);
    virtual void append_line(const char *new_line = 0) { line( lines_.size(), new_line ); };
    virtual void append_lines(const char *new_lines) { lines( lines_.size(), new_lines ); };
    virtual const char *line(int i) const;
    virtual int line_count() const;
    virtual void reset_line_count(int num_lines = 0);
    virtual void is(const Modifier *);
    virtual void display_data(ostream & );
public:
    virtual const char *class_name() const { return "Text"; }
}
```

#### DESCRIPTION

This class is used if the application programmer needs to accept input from the user in the form of multiple line free text.

#### MEMBER FUNCTIONS

DUI\_Text::DUI\_Text()

Description: Empty constructor. returns: void

DUI\_Text::DUI\_Text(const char \*name, const char \*sample\_value)

Description: Constructor accepting a name and initial value. returns: void

DUI(1)      Last change: Tue Jan 4 16:20:18 1994      1

DUI\_Text(1)                      Gatec Manual                      DUI\_Text(1)

DUI\_Text::~~DUI\_Text()

Description: Destructor deletes all lines. returns:  
void

DUI\_Text::line(int i)

Description: Accesser function. returns: value of line i or "";

DUI\_Text::remove\_lines(int first, int last)

Description: removes lines "first"-"last" unless "last" == 0 (default) in which case it will only remove line(first). returns: void

DUI\_Text::line\_count()

Description: Accessor funtion. returns: the number of lines.

DUI\_Text::reset\_line\_count( int num\_lines )

Description: Sets the number of lines. returns: void

DUI\_Text::insert\_line( int i, const char \*new\_line )

Description: Inserts a line at index "i". returns:  
void

DUI\_Text::line(int i, const char \*new\_line)

Description: Sets the value of line i to new\_value. returns: void

DUI\_Text::lines(int i, const char \*new\_lines)

Description: Sets the value of several lines (starting with line i) Multiple lines in new\_values should be separated by newline characters. returns: void

DUI\_Text::is( const Modifier \*modifier )

Description: Allows the applciation to specify a Modifier for the text data (i.e. Lower\_Case, Truncated, Left\_Justified). returns: void

DUI\_Text::display\_data( ostream &out )

Description: Prints a simple ascii representation of DUI\_Text onto stream "out". returns: void

FILES

DUI\_Text.C DUI\_Text.H

### 2.1.8.21 DUI\_Toggle

#### NAME

DUI\_Toggle - Provides support for on-off switch.

#### SYNOPSIS

```
#include "DUI_Toggle.H"

class DUI_Toggle: public DUI_Component {

communication_decls(DUI_Toggle)
private:
    boolean    value_;
protected:
    DUI_Toggle();
public:
    virtual ~DUI_Toggle();
    DUI_Toggle(const char *name, boolean value = 0);
    boolean value() const;
    void value(int );
    enum {OFF, ON, TOGGLE};
    virtual void display_data(ostream & );
public:
    virtual const char *class_name() const { return "Toggle";
}
}
```

#### DESCRIPTION

This class is used when the application programmer wants the user to specify one of two states (on or off) for a value. (e.g. check box).

#### MEMBER FUNCTIONS

DUI\_Toggle::DUI\_Toggle()

Description: Empty Constructor. returns: void

DUI\_Toggle::DUI\_Toggle(const char \*name, boolean value)

Description: Constructor accepting a name and initial value.  
returns: void

DUI\_Toggle::~~DUI\_Toggle()

Description: Destructor. Does nothing. returns: void

DUI\_Toggle::value()

Description: Returns the value of this DUI\_Toggle. returns:  
DUI\_Toggle::ON or DUI\_Toggle::OFF.

DUI(1)      Last change: Tue Jan 4 16:20:16 1994      1

DUI\_Toggle(1)      Gatec Manual      DUI\_Toggle(1)

    DUI\_Toggle::value( int new\_value )

    Description: Sets (of toggles) the value of this DUI\_Toggle.  
    returns: void

    DUI\_Toggle::display\_data( ostream &out )

    Description: Print a simple ASCII representation of this object  
    onto stream "out". returns: void

## FILES

DUI\_Toggle.C DUI\_Toggle.H

### 2.1.8.22 DUI\_View

#### NAME

DUI\_View - Provides support for a view(screen) which contains other objects.

#### SYNOPSIS

```
#include "DUI_View.H"
```

```
class DUI_View: public DUI_Widget {
```

```
communication_decls(DUI_View)
```

```
private:
```

```
    DUI_Component*  component_;  
    DUI_Command*    command_;  
    DUI_Command*    choice_;  
    static List_of(DUI_View) *waiting_list;  
    DUI_View*  previous_view_;
```

```
public:
```

```
    virtual ~DUI_View();  
    virtual void component(DUI_Component *);  
    virtual void command(DUI_Command *);  
    virtual DUI_Component *component() const;  
    virtual DUI_Command *command() const;  
    void choose(DUI_Command *choice);  
    DUI_Command *choice();  
    virtual void update(int changed = 1);  
    virtual void display();  
    virtual void display_data(ostream & );
```

```
protected:
```

```
    friend class Session;  
    virtual void receive();
```

```
protected:
```

```
    virtual void send();  
    DUI_View();
```

```
    DUI_View(const char *label, DUI_Component *component = 0, DUI_Command *command = 0);
```

```
public:
```

```
    virtual const char *class_name() const { return "View"; }
```

```
}
```

#### DESCRIPTION

This class is the base class for DUI\_Form(which see) which is the application programmers interface for creating views(forms or screens). A view contains one component and one command. In all accept the simplest forms the component will be a DUI\_Group(which see) which will contain all the other widgets the application programmer wants to appear on the screen. The

command as well will most likely be a composite command (or command group see `DUI_Command`) which will contain all the view-level commands that the programmer wants to appear on the screen.

## MEMBER FUNCTIONS

`DUI_View::DUI_View()`

Description: Empty Constructor. returns: void

`DUI_View::DUI_View( const char * label, DUI_Component * component, DUI_Command * command )` Description: Constructor accepting a name, a component and a command. returns: void

`DUI_View::~~DUI_View()`

Description: Desctructor. Deletes the component and command. returns: void

`DUI_Command *DUI_View::command()`

Description: Accesser function. returns: `DUI_Command *` the views command.

`DUI_Component *DUI_View::component()`

Description: Accesser function. returns: `DUI_Component *` the view's component.

`void DUI_View:: command( DUI_Command *command)`

Description: Sets the view's command. returns: void

`void DUI_View::component( DUI_Component *component)`

Description: Sets the view's component. returns: void

`void DUI_View::choose( DUI_Command *choice )`

Description: This function sets the view's `choice_` which is the command that is currently chosen at the view level. returns: void

`DUI_Command * DUI_View::choice()`

Description: Accesser function. returns: `DUI_Command *`, the currently chosen command.

`void DUI_View::update( int changed )`

Description: Just sets `need_to_update()` flag, but doesn't send. The `need_to_update` flag is used when the decision is being made about what to send from the application to the client and vice-versa. (see `DUI` and `Communication_Object`). returns: void

`void DUI_View::send()`

Description: This function makes sure commands are pointing to view before sending. It is overloads `Communication_Object::send()` (which see). returns: void

`void DUI_View::display()`

Description: Displays this view to the user. This function actually just places the view on a waiting list if the view is not the first to be sent during this Call-back cycle (see `DUI`). returns:

void

DUI\_View::display\_data( ostream &out )

Description: Print a simple ascii representation of this object onto stream "out". returns: void

## FILES

DUI\_View.C DUI\_View.H



### 2.1.8.23 DUI\_Widget

#### NAME

DUI\_Widget - Base class for all DUI toolkit elements.

#### SYNOPSIS

```
#include "DUI_Widget.H"

class DUI_Widget: public Communication_Object {

communication_decls(DUI_Widget)
private:
    STRING*    name_;
public:
    virtual ~DUI_Widget();
    virtual const char *name() const;
    virtual void name(const char *name);
    virtual void display_data(ostream & out);
protected:
    DUI_Widget();
    DUI_Widget(const char *name);
    virtual void client_construct() {};
    virtual void client_destruct() {};
public:
    virtual const char *class_name() const { return "Widget";
    }
}
```

#### DESCRIPTION

This class is never instantiated directly, but is the base class for all the DUI widgets and more specifically for DUI\_Component(which see) and DUI\_View (which see). All it contains is a name which is something that is shared by all DUI toolkit elements.

#### MEMBER FUNCTIONS

DUI\_Widget::DUI\_Widget()

Description: Empty constructor. returns: void

DUI\_Widget::DUI\_Widget(const char \*name )

Description: Constructor accepting a name as an argument.  
returns: void

DUI\_Widget::~~DUI\_Widget()

Description: Destructor. Deletes name. returns: void

const char \*DUI\_Widget::name()

Description: Accesser function. returns: const char \* the name of the widget.

void DUI\_Widget::name( const char \*name )

Description: Sets the name of the widget. returns: void

DUI\_Widget::display\_data( ostream &out )

Description: Print a simple ascii representation of this object onto stream "out". returns: void

## FILES

DUI\_Widget.C DUI\_Widget.H

### 2.1.8.24 Date

#### NAME

Date - Date constraint class.

#### SYNOPSIS

```
#include "Date.H"

class Date: public Constraint {

communication_decls(Date)
public:
    Date();
    virtual ~Date();
    virtual const char *invalid(const char *string) const;
    static const char *format();
public:
    virtual const char *class_name() const { return "Date"; }
}
```

#### DESCRIPTION

This class is used to constrain values to the following form:  
{0-31} {JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC} {0-9}{0-9}

#### MEMBER FUNCTIONS

Date::Date()  
Description: Constructor for Date. returns: void

Date::~~Date()  
Description: Destructor. Does nothing. returns: void

Date::invalid( const char \*date )  
Description: Error message returned which describes Date format. returns: char \*, error message or 0.

const char \*Date::format()  
Description: Accessor function. Provides an oracle date string representing desired format. returns: char \*, the oracle date string.

#### FILES

Date.C Date.H

### 2.1.8.25 Filebuf\_With\_Audit

#### NAME

Filebuf\_With\_Audit - A filebuf derivative that writes its data to a log as well.

#### SYNOPSIS

```
#include "Filebuf_With_Audit.H"

class Filebuf_With_Audit: public filebuf { public:
    Filebuf_With_Audit();
    Filebuf_With_Audit(int primaryfd, int auditfd);
    virtual ~Filebuf_With_Audit();
    Filebuf_With_Audit *attach_audit_fd(int fd);
protected:
private:
    int audit_fd();
    virtual int overflow(int c = EOF);
    int audit_fd_;
}
```

#### DESCRIPTION

This is a debug filebuf class that writes all its output to a separate file descriptor assigned by the programmer for logging purposes as well as its original file descriptor.

#### MEMBER FUNCTIONS

Filebuf\_With\_Audit \*Filebuf\_With\_Audit::attach\_audit\_fd(int fd)

Description: Function to attach an audit file descriptor. returns: Filebuf\_With\_Audit \*, this.

Filebuf\_With\_Audit::Filebuf\_With\_Audit()

Description: Empty Constructor. returns: void

auditfd)

Filebuf\_With\_Audit::Filebuf\_With\_Audit(int primaryfd, int

Description: Constructor accepting primary file descriptor as well as audit file descriptor as arguments. returns: void

int Filebuf\_With\_Audit::audit\_fd()

Description: Accessor function. returns: int, audit file descriptor.

Filebuf\_With\_Audit::~~Filebuf\_With\_Audit()

Description: Destructor for Filebuf\_With\_Audit. Does NOT close the audit file descriptor. returns: void

int Filebuf\_With\_Audit::overflow(int c )

Description: Overflow() calls filebuf::overflow() after writing buffer to audit\_fd\_. returns: int, return value of filebuf::overflow().

## FILES

Filebuf\_With\_Audit.C Filebuf\_With\_Audit.H

### 2.1.8.26 Integer

#### NAME

Integer - Constrains a value to integer format.

#### SYNOPSIS

```
#include "Integer.H"

class Integer: public Constraint {

communication_decls(Integer)
private:
    int  valid_sign_;
public:
    Integer(int valid_sign = 1);
    virtual ~Integer();
    enum {POSITIVE_ONLY, POSITIVE_OR_NEGATIVE};
    virtual const char *invalid(const char *string) const;
public:
    virtual const char *class_name() const { return "Integer"; }
}
```

#### DESCRIPTION

This class constrains value to the following forms:

with POSITIVE\_ONLY set:

any number of digits preceeded by an optional plus sign.

with POSITIVE\_OR\_NEGATIVE set:

any number of digits preceeded by an optional plus or minus sign.

#### MEMBER FUNCTIONS

Integer::Integer(int valid\_sign)

Description: Empty constructor. returns: void

Integer::~~Integer()

Description: Destructor. does nothing returns: void

const char \*Integer::invalid(const char \*string)

Description: Returns error message if string is not Integer.  
returns: char \*, error message, or 0.

#### FILES

Integer.C Integer.H

### 2.1.8.27 Justified

#### NAME

Justified - Centers a value.

#### SYNOPSIS

```
#include "Justified.H"

class Justified: public Modifier {

communication_decls(Justified)
private:
    int length_;
public:
    Justified(int length = 0);
    virtual ~Justified();
    virtual void modify(STRING & string) const;
public:
    virtual const char *class_name() const { return "Justified"; }
}
```

#### DESCRIPTION

This class is derived from Modifier and is used to center a value.

#### MEMBER FUNCTIONS

Justified::Justified(int length)

Description: Constructor accepting a length which is used to gauge the centering. returns: void

Justified::~~Justified()

Description: Destructor. does nothing. returns: void

Justified::modify(STRING & string)

Description: Center justify string using length specified in constructor. returns: void

#### FILES

Justified.C Justified.H

### 2.1.8.28 Left\_Justified

#### NAME

Left\_Justified - Left justifies a value.

#### SYNOPSIS

```
#include "Left_Justified.H"

class Left_Justified: public Modifier {

communication_decls(Left_Justified)
private:
    int length_;
public:
    Left_Justified(int length = 0);
    virtual ~Left_Justified();
    virtual void modify(STRING & string) const;
public:
    virtual const char *class_name() const { return "Left_Justified"; }
}
```

#### DESCRIPTION

This class is used to left justify a value.

#### MEMBER FUNCTIONS

Left\_Justified::Left\_Justified(int length)

Description: Constructor accepting length used gauge justification. returns: void

Left\_Justified::~~Left\_Justified()

Description: Destructor. Does nothing. returns: void

Left\_Justified::modify(STRING & string)

Description: Left justify string using length specified in constructor. returns:

#### FILES

Left\_Justified.C Left\_Justified.H



### 2.1.8.29 Lower\_Case

#### NAME

Lower\_Case - Changes value to lower case.

#### SYNOPSIS

```
#include "Lower_Case.H"

class Lower_Case: public Modifier {

communication_decls(Lower_Case)
public:
    Lower_Case();
    virtual ~Lower_Case();
    virtual void modify(STRING & string) const;
public:
    virtual const char *class_name() const { return "Lower_Case"; }
}
```

#### DESCRIPTION

This class is used to shift a string to all lower case.

#### MEMBER FUNCTIONS

Lower\_Case::Lower\_Case()

Description: Empty constructor for Lower\_Case Modifier. returns:  
void

Lower\_Case::~~Lower\_Case()

Description: Destructor. Does nothing. returns: void

void Lower\_Case::modify(STRING & string)

Description: Convert string to lower case. returns:

#### FILES

Lower\_Case.C Lower\_Case.H

### 2.1.8.30 Mandatory

#### NAME

Mandatory - Constrains a value to have a length greater than 0.

#### SYNOPSIS

```
#include "Mandatory.H"
```

```
class Mandatory: public Constraint {
```

```
communication_decls(Mandatory)
```

```
public:
```

```
    Mandatory();
```

```
    virtual ~Mandatory();
```

```
    virtual const char *invalid(const char *string) const;
```

```
public:
```

```
    virtual const char *class_name() const { return "Mandatory"; }
```

```
}
```

#### DESCRIPTION

This class allows the programmer to require input from the user.

#### MEMBER FUNCTIONS

Mandatory::Mandatory()

Description: Empty constructor. returns: void

Mandatory::~~Mandatory()

Description: Destructor. void returns:

Mandatory::invalid(const char \*string)

Description: Returns err message if string is empty. returns: char \*, the error message or 0.

#### FILES

Mandatory.C Mandatory.H

### 2.1.8.31 Military\_Date

NAME

Military\_Date - Military date constraint class.

SYNOPSIS

```
#include "Military_Date.H"
```

```
class Military_Date: public Constraint {
```

```
communication_decls(Military_Date)
```

```
public:
```

```
    Military_Date();
```

```
    virtual ~Military_Date();
```

```
    virtual const char *invalid(const char *string) const;
```

```
public:
```

```
    virtual const char *class_name() const { return "Military_Date"; }
```

```
    }
```

DESCRIPTION

This class is used to constrain values to the following form:

{0-9}{0-9} {JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG,  
SEP, OCT, NOV, DEC} {0-31}

MEMBER FUNCTIONS

Description:

Constructor for Military\_Date. returns: void

Military\_Date::~~Military\_Date()

Description: Destructor. Does nothing. returns: void

Military\_Date::invalid( const char \*date )

Description: Error message returned which describes  
Military\_Date format. returns: char \*,the error message or 0.

FILES

Military\_Date.C Military\_Date.H

### 2.1.8.32 Modifier

#### NAME

Modifier - Base class for all DUI modifiers.

#### SYNOPSIS

```
#include "Modifier.H"
```

```
class Modifier: public Communication_Object {  
  
communication_decls(Modifier)  
public:  
    virtual ~Modifier();  
    virtual void modify(STRING & string) const;  
protected:  
    Modifier();  
public:  
    virtual const char *class_name() const { return "Modifier"; }  
}
```

#### DESCRIPTION

This class provides a base for the DUI Modifiers such as Left\_Justify (which see). Modifiers can be attached to DUI\_Field's(which see), DUI\_Text's(which see) and Table\_Column's(which see). A modifier is applied to the value before Constraint's(which see) are checked. A modifier modifies the value according to its definition.

#### MEMBER FUNCTIONS

Modifier::Modifier()

Description: Empty constructor for Modifier. returns:  
void

Modifier::~~Modifier()

Description: Destructor. Does nothing. returns: void

void Modifier::modify( STRING & )

Description: This function must be overloaded by derived classes. It should modify the given value according to its definition. returns: void

#### FILES

Modifier.C Modifier.H

### 2.1.8.33 Numeric

#### NAME

Numeric - Numeric constraint for a value.

#### SYNOPSIS

```
#include "Numeric.H"

class Numeric: public Constraint {

communication_decls(Numeric)
public:
    Numeric();
    virtual ~Numeric();
    virtual const char *invalid(const char *string) const;
public:
    virtual const char *class_name() const { return "Numeric"; }
}
```

#### DESCRIPTION

This class is used to ensure a value is in a format recognized by strtod()(which see).

#### MEMBER FUNCTIONS

Numeric::Numeric()

Description: Empty constructor. returns: void

Numeric::~~Numeric()

Description: Destructor. Does nothing. returns: void

const char \*Numeric::invalid(const char \*string)

Description: Returns error message if string is not numeric.  
returns: char \*, the error message or 0.

#### FILES

Numeric.C Numeric.H

### 2.1.8.34 Precision

#### NAME

Precision - Modifier that sets precision of a numeric string.

#### SYNOPSIS

```
#include "Precision.H"
```

```
class Precision: public Modifier {
```

```
communication_decls(Precision)
```

```
private:
```

```
    int left_of_decimal_;
```

```
    int right_of_decimal_;
```

```
public:
```

```
    Precision(int left_of_decimal = 0, int right_of_decimal = 0);
```

```
    virtual ~Precision();
```

```
    virtual void modify(STRING & string) const;
```

```
public:
```

```
    virtual const char *class_name() const { return "Precision"; }
```

```
}
```

#### DESCRIPTION

This class is used if the programmer wishes to display a number to a specific precision.

#### MEMBER FUNCTIONS

```
Precision::Precision(int left_of_decimal, int right_of_decimal )
```

Description: Constructor accepts two arguments, number of places to the left of the decimal desired, and number of places to the right of the decimal desired. A 0 (default) means give only significant places. returns: void

```
Precision::~~Precision()
```

Description: Destructor. Does nothing. returns: void

```
void Precision::modify(STRING & string)
```

Description: Modify() is called to apply the precision to the given string. It uses lengths passed to constructor to figure the precision. It will only use the left-of-decimal length if it greater than the length of the string, in which case it will pad with 0's. returns: void

DUI(1)      Last change: Tue Jan 4 16:21:01 1994      1

Precision(1)      Gatec Manual      Precision(1)

#### FILES

Precision.C Precision.H  
**2.1.8.35 Regular\_Expression**

NAME

Regular\_Expression - Provides support for a regular expression constraint.

SYNOPSIS

```
#include "Regular_Expression.H"

class Regular_Expression: public Constraint {

communication_decls(Regular_Expression)
private:
    STRING*    expression_;
    Regular_Expression();
public:
    Regular_Expression(const char *expression);
    virtual ~Regular_Expression();
    virtual const char *invalid(const char *string) const;
    virtual const char *expression() const;
    virtual void expression(const char *exp);
public:
    virtual const char *class_name() const { return "Regular_Expression"; }
}
```

DESCRIPTION

This class is used to apply a regular expression to a value as its constraint. The draw back to using a regular expression is that the error message returned is only informative to someone who understands regular expressions.

MEMBER FUNCTIONS

Regular\_Expression::Regular\_Expression(const char \*exp)  
Description: Constructor accepting a regular expression as an argument. returns: void

Regular\_Expression::~~Regular\_Expression()  
Description: Destructor. Deletes regular expression. returns: void

Regular\_Expression::expression( const char \*exp )  
Description: Sets the regular expression pattern. returns: void

Regular\_Expression::expression()  
Description: Returns the regular expression pattern. returns: char \*, the regular expression.

DUI(1)      Last change: Tue Jan 4 16:21:00 1994      1

Regular\_Expression(1)      Gatec      Manual  
Regular\_Expression(1)

Regular\_Expression::invalid ( const char \*string )

Description: returns 0 if string matches the regular expression  
'expression\_' returns an informative message otherwise returns:  
char \*, the message or 0.

## FILES

Regular\_Expression.C Regular\_Expression.H



### 2.1.8.36 Right\_Justified

#### NAME

Right\_Justified - Right justifies a value.

#### SYNOPSIS

```
#include "Right_Justified.H"

class Right_Justified: public Modifier {

communication_decls(Right_Justified)
private:
    int length_;
public:
    Right_Justified(int length = 0);
    virtual ~Right_Justified();
    virtual void modify(STRING & string) const;
public:
    virtual const char *class_name() const { return "Right_Justified"; }
}
```

#### DESCRIPTION

This class is used to right justify a value. It is derived from Modifier(which see).

#### MEMBER FUNCTIONS

Right\_Justified::Right\_Justified(int length)

Description: Constructor accepting the length used to gauge justification. returns: void

Right\_Justified::~~Right\_Justified()

Description: Destructor. Does nothing. returns: void

Right\_Justified::modify(STRING & string)

Description: Right justify string using length specified in constructor. returns: void

#### FILES

Right\_Justified.C Right\_Justified.H

### 2.1.8.37 STRING

#### NAME

STRING - A generic string class.

#### SYNOPSIS

```
#include "STRING.H"
```

```
class STRING: public Communication_Object {
```

```
communication_decls(STRING)
```

```
protected:
```

```
    friend ostream &operator << ( ostream &, STRING & );
```

```
    friend istream &operator >> ( istream &, STRING & );
```

```
    char *value_;
```

```
    int length_;
```

```
    int size_;
```

```
    static STRING *buf_;
```

```
    STRING();
```

```
private:
```

```
    void resize(int size);
```

```
    void set(const char *value, int len);
```

```
public:
```

```
    static STRING &buf();
```

```
    STRING(int size);
```

```
    STRING(STRING & );
```

```
    STRING(const char *str);
```

```
    STRING(const char *str, int length);
```

```
    virtual ~STRING();
```

```
    char *value() { return value_; };
```

```
    operator char *() { return value_; };
```

```
    int length() { return length_; };
```

```
    STRING &operator = (STRING & str);
```

```
    STRING &operator +=(STRING & str);
```

```
    STRING &operator = (const char *str);
```

```
    STRING &operator +=(const char *str);
```

```
    STRING &operator +=(char );
```

```
    boolean operator ==(STRING & str) { return (strcmp(value_, str.value_) == 0); };
```

```
    boolean operator !=(STRING & str) { return (strcmp(value_, str.value_) != 0); };
```

```
    boolean operator > (STRING & str) { return (strcmp(value_, str.value_) > 0); };
```

```
    boolean operator >=(STRING & str) { return (strcmp(value_, str.value_) >= 0); };
```

```
    boolean operator < (STRING & str) { return (strcmp(value_, str.value_) < 0); };
```

```
    boolean operator <=(STRING & str) { return (strcmp(value_, str.value_) <= 0); };
```

```
    boolean operator ==(const char *str) { return (strcmp(value_, str ? str : "") == 0); };
```

```
    boolean operator !=(const char *str) { return (strcmp(value_, str ? str : "") != 0); };
```

```
    char operator[](int index);
```

```
    boolean convert(int & );
```

```
    boolean convert(long & );
```

```

boolean convert(float & );
boolean convert(double & );
void unjustify();
void center_justify(int length);
void right_justify(int length);
void left_justify(int length);
public:
virtual const char *class_name() const { return "STRING";
}
}

```

## DESCRIPTION

This class is a generic string class which has been made into a Communication\_Object(which see) for use with DUI.

## MEMBER FUNCTIONS

### STRING::STRING()

Description: Private constructor for internal use only. returns: void

### STRING::STRING( int size )

Description: Constructs empty, null-terminated String of length size. returns: void

### STRING::STRING(const char \*str)

Description: Constructor that copies a NULL-terminated array of char. returns: void

### STRING::STRING(const char \*str, int length)

Description: Constructor that copies a non NULL-terminated array of char using length. returns: void

### STRING::~~STRING()

Description: Destructor for STRING. returns: void

### STRING::set( const char \*value, int size )

Description: sets value\_ to value and length\_ to length (growing String if needed) and NULL-terminates value\_ returns: void

### STRING::resize( int size )

Description: Resize value\_ if needed. returns: void

### STRING& STRING::operator += (STRING & str)

Description: Concatenate str to end of STRING. returns: the STRING&.

### STRING& STRING::operator += ( const char \* chars )

Description: Concatenates char \* chars to end of string. returns: this String&.

STRING& STRING::operator += ( char c )

Description: Concatenates char c to end of string. returns: this String &.

STRING& STRING::operator = (STRING & str)

Description: Assignment operator for STRING (from STRING). returns: the STRING &.

STRING& STRING::operator = (const char \* str)

Description: Assignment operator for STRING (from char \*). returns: the STRING &.

STRING::operator[](int index)

Description: Operator [n] returns the nth char in STRING. returns: char the indexed character.

STRING::convert(int &num)

Description: Converts string to int. returns: 1 if successful, 0 otherwise

STRING::convert(long &num)

Description: Converts string to long. returns: 1 if successful, 0 otherwise

STRING::convert(float &num)

Description: Converts string to float. returns: 1 if successful, 0 otherwise

STRING::convert(double &num)

Description: Converts string to double. returns: 1 if successful, 0 otherwise

DUI(1)      Last change: Tue Jan 4 16:20:56 1994      3

STRING(1)      Gatec Manual      STRING(1)

STRING::co\_print( ostream &out )

Description: This code is generated for other communication objects. It was too difficult for STRING so it is written by hand. This function writes out the string object onto stream. returns: void

STRING::co\_parse( istream &in )

Description: This code is auto-generated for other communication objects. It was too difficult for STRING because we needed to use length to allocate enough space. This function reads in a STRING off the stream. returns: void

STRING::unjustify()

Description: strips leading and trailing spaces. returns: void

STRING::left\_justify( int len )

Description: removes trailing spaces, pads with leading spaces.  
returns: void

STRING::center\_justify( int len )

Description: makes number trailing spaces = number leading spaces. returns: void

STRING::right\_justify( int len )

Description: removes leading spaces, pads with trailing spaces.  
returns: void

STRING::buf()

Description: This function allows access to static buf\_. returns:  
String & buf\_.

FILES

STRING.C STRING.H

### 2.1.8.38 Session

#### NAME

Session - Provides support for opening up a DUI communications session.

#### SYNOPSIS

```
#include "Session.H"
```

```
class Session { protected:
    Session( char *programe );
    ~Session();
    int      status;
    int      running;
    istream*  inchannel;
    ostream*  outchannel;
    AppControl* thisapp;
    ConfigInfo* configuration;
    ofstream*  log_;
    static Session *instance();
    static Session *instance_;
public:
    static void send(Communication_Object* );
    static void run();
    static int inerror();
    static int end();
    static ofstream& log();
    static void warning( const char *c );
    static void debug( const char *c );
};
/* * Client_Session class definition. * */

class Client_Session: public Session { private:
    Client_Session(char *programe): Session(programe) { };
    ~Client_Session() { };
public:
    static int begin(char *appname, void (*efp)()=0);
};
/* * Server_Session class definition. * */

class Server_Session: public Session { private:
    Server_Session(char *programe): Session(programe) { };
    ~Server_Session() { };
public:
    static int begin(char *appname, void (*efp)()=0);
};
/* * Application_Session class definition. * */
```

```

class Application_Session: public Session { private:
    Application_Session(char *progrname): Session(progrname)
    {};
    ~Application_Session() {};
public:
    static int begin(char *appname, void (*efp)()=0);
}

```

## DESCRIPTION

The class Session is used as a base class for Client\_Session, Server\_Session and Application\_Session. These classes differ only in their definitions of the begin() function which is called to establish a connection when the client, server, or application starts up. See DUI for a more detailed description of the DUI communications paradigm.

## MEMBER FUNCTIONS

inline Session \*Session::instance()

Description: There is only one instance of a Session in a program. This function gives the user access to that instance. returns: Session \*, the Session.

Session::Session(char \*appname)

Description: Constructor accepting an application name as argument. It opens a log file in the form: dui.log.<application name>.<uid>. returns: void

void Session::send(Communication\_Object\* cobject)

Description: This function calls the << operator on the passed Communication\_Object through the outchannel established by the begin() function. returns: void

Session::~~Session()

Description: Destructor. Deletes inchannel, outchannel, thisapp, configuration. returns: void

int Session::end()

Description: This function is called to close a communications session. It sends the AppControl(which see) object to the other side of the session after setting its end flag and then calls receive on it. This should end the application on this side as well. returns: -1 if it returns at all, it should not return.

int Session::inerror()

Description: Status function. returns: 1 if error, 0 otherwise.

void Session::run()

Description: This function is called by DUI\_View::send()

when it sends the first view. It goes into a loop stopped only by a failure on the communications line. This basically sets up an "event" loop where by actions are initiated when data is received from the client or application. The loop is terminated when Session::end() is called because exit() is called by the receive function for AppControl which is executed by Session::end(). The loop is protected against re-entrance by a "running" flag. returns: void

ofstream& Session::log( )

Description: Accesser function. returns: ofstream &, the log file.

void Session::warning( const char \*c )

Description: Writes out the char \* argument to the log file flagged as a warning. returns: void

void Session::debug( const char \*c )

Description: Writes out the char \* argument to the log file flagged as debug. returns: void

int Client\_Session::begin(char \*appname, void (\*efp)

Description: Begin() function for Client\_Session sub-class. This function reads in the configuration information and based upon what it says establishes the communications channels, sends the AppControl object for this client, and waits for the AppControl object to be sent back from the application acknowledging proper start up. returns: int, success or failure.

int Server\_Session::begin(char \*appname, void (\*efp)

Description: Begin() function for Server\_Session sub-class. The server (which is a specialized application) starts up by open stdin and stdout as its channels, reading in the AppControl which it expects on its inchannel and executing that AppControl object. This should start up the application. returns: int -1 if it returns which it shouldn't.

int Application\_Session::begin(char \*appname, void (\*efp)

Description: Begin() function for Application\_Session subclass. The application starts up by opening up stdin and stdout as its inchannel and outchannel and sending its AppControl as confirmation that it started up successfully. returns: 0 always.

## FILES

Session.C Session.H



### 2.1.8.39 SocketBuf

#### NAME

SocketBuf - streambuf derivative for sockets.

#### SYNOPSIS

```
#include "SocketBuf.H"
```

```
class SocketBuf: public ChannelBuf { public:
    SocketBuf();
    virtual ~SocketBuf();
    virtual int connect(char *host, int port);
    SocketBuf *attach(int fd);
    SocketBuf *attach_audit_fd(int fd);
protected:
private:
    const char *host();
    int port();
    int fd();
    int audit_fd() { return _audit_fd; };
    int opened();
    int nonblocking();
    int nonblocking(int);
    virtual int connect();
    SocketBuf *accept(int & fd);
    SocketBuf *verbose(int );
    virtual int disconnect();
    virtual int overflow(int c = EOF);
    virtual int underflow();
    virtual int sync();
    virtual int doallocate();
    void error(const char *);
    void sys_error(const char *);
    char *_host;
    int _port;
    int _fd;
    int _audit_fd;
    int _opened;
    int _close;
    int _nonblocking;
    int _verbose;
}
```

#### DESCRIPTION

This class implements a streambuf using a socket as the sink and source for the data.

## MEMBER FUNCTIONS

SocketBuf \*SocketBuf::accept(int & fd)

Description: This function is used to accept an incoming socket connection request. returns: SocketBuf\*, this.

SocketBuf \*SocketBuf::attach(int fd)

Description: Sets the file descriptor used on reads and writes to the passed fd if there is not one set already. returns: SocketBuf \*, this if successful or 0 if not.

SocketBuf \*SocketBuf::attach\_audit\_fd(int fd)

Description: Attach a file descriptor to send audit to. The audit file descriptor is written to before any writes to the primary file descriptor. returns: SocketBuf \*, this.

int SocketBuf::connect(char \*host, int port)

Description: This function attempts to establish a connection to the passed "host" and "port" number. Host can be either an ip address or a host name. It attempts to connect() 4 times. returns: 1 if successful, 0 otherwise.

int SocketBuf::connect()

Description: Default connect for testing purposes. Should be removed. returns: 1 if successful, 0 otherwise.

int SocketBuf::disconnect()

Description: Attempts to close the socket. returns: 1 if successful, 0 otherwise.

SocketBuf \*SocketBuf::verbose(int verbose )

Description: Sets verbose flag. returns: SocketBuf\*, this.

SocketBuf::SocketBuf()

Description: Empty constructor. Initializes data members. returns: void

const char \*SocketBuf::host()

Description: Accessor function. returns: const char \*, host name or ip address.

int SocketBuf::fd()

Description: Accessor function. returns: int file descriptor, -1 if none.

int SocketBuf::nonblocking()

Description: Accessor function. returns: blocking mode.

int SocketBuf::nonblocking(int nonblocking)

Description: Sets blocking mode. Read will not return until

something is on the socket or there is an error, if nonblocking is set to no(0). returns: current blocking mode.

int SocketBuf::opened()

Description: Status function. returns: 0 if not opened, 1 otherwise.

int SocketBuf::port()

Description: Accessor function. returns: current port number or 0.

void SocketBuf::error(const char \*msg)

Description: If verbose is turned on it writes the error message to cerr. returns: void

void SocketBuf::sys\_error(const char \*msg)

Description: Uses perror() to output the message passed as well as the last error that occurred on a system call. returns: void

SocketBuf::~~SocketBuf()

Description: Destructor. deletes buffer space and calls disconnect(). returns: void

int SocketBuf::doallocate()

Description: Allocates buffer space. Allocates separate buffer space for put and get. returns: 0 if successful EOF if not.

int SocketBuf::overflow(int c )

Description: Overflow() for a socket sink. Writes to the audit file descriptor. as well as to the primary file descriptor(socket). returns: int number of characters written.

int SocketBuf::sync()

Description: This function resets the get buffer(everything not read is lost.) and flushes the put buffer to the socket(writes everything out). returns: int overflow() return value. (the number of chars written).

int SocketBuf::underflow()

Description: Reads from the source (socket) as much as its buffer will currently hold minus what hasn't been read yet, or whatever read returns on a successful attempt. returns: int next character in buffer or EOF.

## FILES

SocketBuf.C SocketBuf.H

#### 2.1.8.40 Table\_Column

NAME

Table\_Column - Class for dealing with columns in a DUI\_Table.

SYNOPSIS

```
#include "Table_Column.H"

class Table_Column: public Communication_Object {

communication_decls(Table_Column)
private:
    STRING *validation_;
    STRING*   name_;
    List_of(STRING) values;
    List_of(Constraint) constraints;
    List_of(Modifier) modifiers;
    int change_ok_;
public:
    Table_Column(const char *name = "", int num_rows = 1);
    virtual ~Table_Column();
    virtual const char *name() const;
    virtual void name(const char *name);
    virtual void is(const Modifier *);
    virtual void is(const Constraint *);
    virtual int row_count() const;
    virtual void reset_row_count(int num_rows);
    virtual void append_row(const char *value = "");
    virtual void insert_row(int row, const char *value = "");
    virtual void remove_row(int row);
    virtual void value(int row, const char *new_value);
    virtual const char *value(int row) const;
    virtual const char *invalid();
    virtual const char *check_invalid();
    virtual void clear_values();
    virtual int width() const;
    virtual void change_ok(boolean );
    virtual boolean change_ok() const { return change_ok_;
};
public:
    virtual const char *class_name() const { return "Table_Column"; }
}
```

DESCRIPTION

This class is used by DUI\_Table(which see) to deal with columns. This is where actual values are stored for each row. It contains a list of STRING's(which see) which are the column

values for each row. It also contains lists of Modifiers(which see) and Constraints(which see) that are applied to this column across all rows.

## MEMBER FUNCTIONS

Table\_Columnn::Table\_Columnn( const char \*name, int num\_rows )  
Description: Constructor accepting a name and number of rows.  
returns: void

Table\_Columnn::~~Table\_Columnn()  
Description: Destructor. Deletes the name. returns:  
void

Table\_Columnn::name()  
Description: Accesser function. returns: char \*, the name of the column.

Table\_Columnn::name( const char \*name )  
Description: Sets the name of a column. returns: void

Table\_Columnn::is( const Modifier \* modifier )  
Description: Attaches a Modifier to this column. returns: void

Table\_Columnn::is( const Constraint \* constraint )  
Description: Attaches a Constraint to this column. returns: void

Table\_Columnn::row\_count()  
Description: Accesser function. returns: int the number of rows in this column.

Table\_Columnn::reset\_row\_count( int num\_rows )  
Description: Resets the number of rows in this column. Removes the trailing rows. returns: void

Table\_Columnn::append\_row( const char \*new\_value )  
Description: Append a row to this column. returns:  
void

Table\_Columnn::insert\_row( int row, const char \*new\_value )  
Description: Insert a row into this column at "row". returns: void

Table\_Columnn::remove\_row(int row)  
Description: Remove row "row" from this column. returns: void

Table\_Columnn::value( int row, const char \*new\_value )  
Description: Set the value of a row in this column. Applies the modifiers and constraints and sets it only if value complies.  
returns: void

Table\_Columnn::value( int row )

Description: Returns the value of a row in the column. returns: char \*, the value at "row".

Table\_Columnn::invalid()

Description: Returns 0 if the previous new\_value passed to value() was valid otherwise returns an explanation of why new\_value is invalid. returns: char \*, 0 or message.

Table\_Columnn::check\_invalid()

Description: Make sure each value is valid. returns: char \*, summation error message or 0.

Table\_Columnn::clear\_values()

Description: Clears all values in column. returns: void

Table\_Columnn::width()

Description: Returns length of longest STRING in column (name or value). returns: int length.

Table\_Columnn::change\_ok( boolean b )

Description: Sets change flag which is used to check wether this column can be changed or not. returns: void

## FILES

Table\_Columnn.C Table\_Columnn.H

### 2.1.8.41 Truncated

#### NAME

Truncated - Modifier to truncate value.

#### SYNOPSIS

```
#include "Truncated.H"
```

```
class Truncated: public Modifier {  
  
    communication_decls(Truncated)  
private:  
    int length_;  
public:  
    Truncated(int length);  
    virtual ~Truncated();  
    virtual void modify(STRING & string) const;  
protected:  
    Truncated();  
public:  
    virtual const char *class_name() const { return "Truncated"; }  
}
```

#### DESCRIPTION

This modifier is used to truncate a value to a length given in the constructor.

#### MEMBER FUNCTIONS

Truncated::Truncated()

Description: Constructor for Truncated. returns:  
void

Truncated::Truncated(int length)

Description: Constructor accepting a length as an argument which is used to truncate a value. returns: void

Truncated::~~Truncated()

Description: Destructor. Does nothing. returns: void

Truncated::modify( STRING &string )

Description: Truncate string to 'length\_'. returns:  
void

#### FILES

Truncated.C Truncated.H

### 2.1.8.42 Unjustified

#### NAME

Unjustified - Modifier used to strip all leading and trailing blanks.

#### SYNOPSIS

```
#include "Unjustified.H"
```

```
class Unjustified: public Modifier {
```

```
communication_decls(Unjustified)
```

```
public:
```

```
    Unjustified();
```

```
    virtual ~Unjustified();
```

```
    virtual void modify(STRING & string) const;
```

```
public:
```

```
    virtual const char *class_name() const { return "Unjustified"; }
```

```
}
```

#### DESCRIPTION

This class is used to unjustify a value(strip all leading and trailing blanks).

#### MEMBER FUNCTIONS

Unjustified::Unjustified()

Description: Constructor. returns: void

Unjustified::~~Unjustified()

Description: Destructor. Does nothing. returns: void

Unjustified::modify(STRING & string)

Description: Strip all leading and trailing spaces from string.  
returns: void

#### FILES

Unjustified.C Unjustified.H



### 2.1.8.43 Upper\_Case

#### NAME

Upper\_Case - Modifier used to change value to upper-case.

#### SYNOPSIS

```
#include "Upper_Case.H"

class Upper_Case: public Modifier {

communication_decls(Upper_Case)
public:
    Upper_Case();
    virtual ~Upper_Case();
    virtual void modify(STRING & string) const;
public:
    virtual const char *class_name() const { return "Upper_Case"; }
}
```

#### DESCRIPTION

This modifier is used to convert all characters in the value to upper-case. It uses toupper()(which see).

#### MEMBER FUNCTIONS

Upper\_Case::Upper\_Case()

Description: Constructor for Upper\_Case Modifier. returns: void

Upper\_Case::~~Upper\_Case()

Description: Destructor. Does nothing. returns: void

void Upper\_Case::modify(STRING & string)

Description: Convert string to Upper case using toupper().  
returns: void

#### FILES

Upper\_Case.C Upper\_Case.H

---

## 2.2 GATEC Application

---

Gatec.oui is an application that fulfills the user interface requirements for the GATEC project. It allows a user to perform various procurement tasks related to the GATEC system. It uses DUI(see DUI(1)) for its user interface, interacts with a data base through the NARQ(see NARQ) and NORA(see NORA) libraries and produces CDF formatted documents through the CDFDB(see CDFDB) library. It is written in C++. To get a user perspective on the gatec.oui application see *GATEC User's Guide* [REF000]

It is comprised of a number of forms that allow the user to review and edit procurement data in the database and issue electronic documents.

The following sections give a technical overview of the gatec.oui application.

---

### 2.2.1 Class Hierarchy

---

The gatec.oui application has the following class hierarchy, indentation denotes derivation:

(DUI\_Form) - defined in DUI(1)

- Award\_Form
- Compose\_Message\_Form
- Flag\_Selection
- Message\_Form
- Quote\_Abstract\_Form
- RFQ\_Category
- Review\_Quote\_Form
- Review\_RFQ\_Form
- Vendor\_Performance\_Form
- Workload\_Form

All classes are DUI\_Form's except for RFQ\_Category. RFQ\_Category is a class for dealing with the categories displayed on the Workload\_Form.

See the individual documentation on these classes for more details.

---

### 2.2.2 Programming Hints

---

The documentation for the individual form classes should be consulted for the specific function of the gatec.oui application that they fulfill. Also the DUI and NARQ and NORA man pages should be consulted because this will clarify a lot of the code used in gatec.oui.

For the most part gatec.oui is a database access program. So the bulk of the code is querying and updating database tables and filling in and taking values from the fields and texts in the gatec forms. All of the code that does this accessing is in the member functions of the form classes.

---

### 2.2.3 GATEC DUI Source Tree

---

The source for gatec.oui is kept under the DUI(1) source tree in:  
\$CVSROOT/oui/applications/gatec

It depends on the NARQ and NORA libraries being in:  
\$CVSROOT/narqdb/lib

and the cdfdb and DUI libraries being in:  
\$CVSROOT/oui/lib

These must be made before the gatec.oui application can be made. To make the gatec.oui application, cd to its source directory and type:

xmkmf; make depend all

The resulting "gatec.oui" file will be installed in:  
\$CVSROOT/oui/bin

---

### 2.2.4 GATEC Form Classes

---

The gatec,oui forms are comprised of the following classes:

Award\_Form  
Compose\_Message\_Form

Flag\_Selection  
Message\_Form  
Quote\_Abstract\_Form  
RFQ\_Category  
Review\_Quote\_Form  
Review\_RFQ\_Form  
Vendor\_Performance\_Form  
Workload\_Form

### 2.2.4.1 Award\_Form

#### NAME

Award\_Form - This form displays information for the awarding process.

#### SYNOPSIS

```
#include "Award_Form.H"
```

```
class Award_Form : public DUI_Form { public:
    static Award_Form *instance( const char *rfq, const char *rfq_line_item, const char *quote_id,
int category );
    ~Award_Form();
    void load_data( const char *rfq, const char *rfq_line_item, const char *quote_id );
    void cancel_award();
    void cancel_award_dialog();
    void quit_award();
    void check_large_business();
    void get_order_statements();
    int create_cancel_cdf();
protected:
    Award_Form();
private:
    int category_;
    static Award_Form *instance_;
    STRING piin_string;
    void commit_award();
    void clear_data();
    int save_data();
    void construct_database_tables();
    void calculate_data();
    void load_award_data( const char *rfq, const char *rfq_line_item, const char *quote_id );
    char *upload_filename;
#ifdef CDF void generate_850( STRING &, char * = 0, char * = "bcasupload");
#endif

    DUI_Field *award_number;
    DUI_Field *rfq_number;
    DUI_Field *line_item;
    DUI_Field *contract;
    DUI_Field *date;
    DUI_Toggle *acknowledgement_required;
    DUI_Field *order_statements;
    DUI_Field *quantity;
    DUI_Field *unit;
    DUI_Field *unit_price;
    DUI_Field *transaction_totals;
    DUI_Field *delivery;
```

```

DUI_Field *awardees_name;
DUI_Field *bcas_vendor_code;
DUI_Field *do_rating;
DUI_Field *bsp;
DUI_Field *fob_point;
DUI_Field *variation;
DUI_Field *discount_percent;
DUI_Field *discount_due_days;
DUI_Field *discount_net_due_days;
DUI_Field *negotiation_authority;
DUI_Field *competition_code;
DUI_Field *confirm;
    DUI_Command *award_cmds;
DUI_Command *view_cmds;
DUI_Group *alb_group;
DUI_Group *alb_shared_group;
DUI_Selection *alb_reason_selection;
Callback *quit_award_callback;
Callback *commit_alb_callback;
    Callback *commit_award_callback;
DUI_Text *order_text;
    DUI_Group *cancel_group;
DUI_Group *no_reopen_group;
DUI_Group *cancel_shared_group;
DUI_Selection *cancel_supp_piin;
DUI_Field *cancel_activity_no;
DUI_Field *cancel_cac;
DUI_Selection *cancel_reason;
DUI_Field *cancel_eff_date;
DUI_Selection *cancel_ftd;
DUI_Selection *cancel_no_reopen;
DUI_Toggle *cancel_cont_sign;
DUI_Field *cancel_no_copies;
DUI_Field *cancel_sus_date;
DUI_Toggle *cancel_oe;
DUI_Toggle *cancel_prs_to_cust;
DUI_Toggle *cancel_with_reopen;
DUI_Field *cancel_order_stats;
DUI_Toggle *cancel_spec_contr;
DUI_Text *cancel_reason_text;
Callback *cancel_award_dialog_callback;
Callback *cancel_award_callback;
DUI_Field *cancel_new_rfq_no;
#ifdef NODB    Document *doc;
    QuoteLineItem *quote_li;
    Quote *quote;
    Award *award;
    AwardLineItem *award_li;
    BCASAward *baward;
    QuoteTerms *qt;

```

```

FreeOnBoard *fob;
GSDefaults *gsd;
ISADefaults *isad;
Acquisition *acq;
ReqForQuoteLineItem *rfq_li;
Stmnt *stmt;
#endif }

```

## DESCRIPTION

This form contains the data necessary for a buyer to make or cancel an award using the GATEC system. It accesses the following tables (see NARQ):

Acquisition, Award, AwardLineItem, BCASAward, Document, DocumentAddressee, DocumentSent, FreeOnBoard, GSDefaults, ISADefaults, LineItem, Part, Quote, QuoteLineItem, QuoteTerms, RelatedPaperwork, ReqForQuote, ReqForQuoteLineItem, SADBUs, Ship, Stmnt, SolicitationLineItem, Vadrs, Vendor, NarqUtil, UserManagerDefaults

There is only one instance of an Award\_Form in an application at one time.

## MEMBER FUNCTIONS

Award\_Form::instance( const char \*rfq, const char \*rfq\_line\_item, const char \*quote\_id, int category )

Description: Public access to constructor, provided because there is only one instance. Each time it is called it loads the data associated with its arguments. When the "category" is Awarded the only action allowed is cancelling. returns: Award\_Form \*, the Award\_Form instance.

Award\_Form::Award\_Form( )

Description: Constructor for Award\_Form. All the DUI widgets and callbacks used on the award screen are instantiated here. returns: void

Award\_Form::construct\_database\_tables()

Description: Construct needed database tables, ignore uninteresting columns. All the tables used on the class level are instantiated here. returns: void

Award\_Form::~~Award\_Form()

Description: Destructor for Award\_Form. Deletes call-backs and class level tables. returns: void

Award\_Form::clear\_data()

Description: Clears the values for all the displayed widgets. returns: void

Award\_Form::calculate\_data()  
Description: Calculate values (and hard-coded defaults requested by WPAFB). returns: void

Award\_Form::load\_data( const char \*rfq, const char \*rfq\_line\_item, const char \*quote\_id ) Description: Loads data from database for this rfq and award. returns: void

Award\_Form::load\_award\_data( const char \*rfq, const char \*rfq\_line\_item, const char \*quote\_id ) Description: Loads award data from database (used for reviewing old awards). returns: void

Award\_Form::quit\_award()  
Description: Quits without awarding this one. Resets requested piin so it may be used again. returns: void

Award\_Form::check\_large\_business()  
Description: If award is to a large business, pops up reason for dissolution Dialog, else commits award to database. Also checks the BCAS vendor code to make sure it is 7 characters long displaying an error message if not. returns: void

Award\_Form::cancel\_award\_dialog()  
Description: Cancel an existing award. Displays the cancel award Dialog. returns: void

Award\_Form::cancel\_award()  
Description: Cancel Award. This displays a no reopen Dialog if more information is required else goes ahead and cancels the award. returns: void

Award\_Form::create\_cancel\_cdf()  
Description: Write CDF which will allow gateway script to actually initiate the cancellation or ammendment of the award. After writing the CDF, it will be placed on the bcascancel queue. returns: int, 1 if successful, 0 otherwise.

Award\_Form::get\_order\_statements()  
Description: Lets the user modify the full text of the order statements as well as "care of" information. returns: void

Award\_Form::commit\_award()  
Description: Writes award info to database and loads next RFQ into Review\_RFQ\_Form. returns: void

Award\_Form::save\_data()  
Description: Saves field values to database. returns: int, 1 if success, 0 if commits failed.



\*queueName)  
Award\_Form::generate\_850( STRING &errmsg, char  
\*cdfFilename, char Description: Queries the tables that are needed  
in call to \_850DBtoDCF() but are not populated yet and calls  
aforementioned. returns: void

## FILES

Award.C Award.H

### 2.2.4.2 Compose\_Message\_Form

#### NAME

Compose\_Message\_Form - The form used to compose a message.

#### SYNOPSIS

```
#include "Compose_Message_Form.H"
```

```
class Compose_Message_Form : public DUI_Form { public:
    static Compose_Message_Form *instance(          const char *new_rfq_number,          const
char *new_line_item,          const char *reply_to = 0,          const char *reply_subject = 0,
const char *reply_ref_num = 0,          const char *reply_doc_id = 0          );
    ~Compose_Message_Form();
    void attach_note();
    void send_message();
protected:
    Compose_Message_Form();
private:
    static Compose_Message_Form *instance_;
    const char *reference_number;
    void setup( const char *new_rfq_number, const char *new_line_item, const char *reply_to,
const char *reply_subject, const char *reply_ref_num, const char *reply_doc_id  );
    STRING *reply_document_id;
    void generate_864( const char *st02,          const char *ref02,          const char *mit01,
const char *dtm02 );
    void save_data( const char *status, boolean send_864 = No );
    DUI_Field  *rfq_number;
    DUI_Field  *line_item;
    DUI_Field  *date;
    DUI_Field  *to;
    DUI_Field  *from;
Gatec(2)    Last change: Tue Jan  4 16:20:08 1994          1
```

```
Compose_Message_Form(2)  Gatec Manual  Compose_Message_Form(2)
    DUI_Field  *subject;
    DUI_Text   *message_body;
}
```

#### DESCRIPTION

This form is used to compose an outgoing message to a vendor or attach an internal note to a utn number. It accesses the following tables(see NARQ):

Document, DocumentAddressee, DocumentSent, GSDefaults, ISADefaults, Message, MessageFrom, MessageReference, MessageTextBody, MessageTo, UserManagerDefaults, Vendor,

There is only one instance of a Compose\_Message\_Form in an application at one time.

## MEMBER FUNCTIONS

`Compose_Message_Form::instance( const char *new_rfq_number, const char *new_line_item, const char *new_to, const char *reply_subject, const char *reply_ref_num, const char *reply_document_id )` Description: Public access to constructor, provided because there is only one instance. Data passed is passed on to `setup()`. Which loads information for this usage of instance. returns: `Compose_Message_Form *`, the instance.

`Compose_Message_Form::setup( const char *new_rfq_number, const char *new_line_item, const char *new_to, const char *reply_subject, const char *reply_ref_num, const char *reply_doc_id )` Description: Loads `message_abstract` with data. If message is a reply it loads the original message in marked as reply to text. returns: void

`Compose_Message_Form::Compose_Message_Form()`  
Description: Constructor for `Compose_Message_Form`. All the widgets used on the form are constructed here. returns: void

`Compose_Message_Form::~~Compose_Message_Form()`  
Description: Destructor for `Compose_Message_Form`.

Gatec(2) Last change: Tue Jan 4 16:20:08 1994 2

`Compose_Message_Form(2)` Gatec Manual  
`Compose_Message_Form(2)`  
Deletes `reply_document_id`. and resets the instance variable. returns: void

`Compose_Message_Form::attach_note()`  
Description: Attach message as a buyer note (internal note). returns: void

`Compose_Message_Form::send_message()`  
Description: Save message as Sent message. Calls `save_data()` to save and generate an 864. returns: void

`gen_864 )`  
`Compose_Message_Form::save_data( const char *status, boolean` Description: Saves data to database as an 864. returns: void

`Compose_Message_Form::generate_864( const char *doc_id, const char *utn, const char *mit01, const char *dtm02 )` Description: Generates 864 text. The X12 864 document is generated directly from this function, as opposed

to other areas in the application where calls are made to functions named \_XXXDBtoCDF() where the X's stand for the document type name. returns: void

## FILES

Compose\_Message.C Compose\_Message.H

### 2.2.4.3 Flag\_Selection

NAME

Flag\_Selection - Specialized selection for the Quote\_Abstract\_Form.

SYNOPSIS

```
#include "Flag_Selection.H"
```

```
class Flag_Selection: public DUI_Selection { private:
    List_of(DUI_Toggle) toggles;
    List_of(STRING) flag_values;
public:
    virtual ~Flag_Selection();
    Flag_Selection( const char * name , DUI_Toggle * = 0,          DUI_Toggle * = 0,
    DUI_Toggle * = 0, DUI_Toggle * = 0,          DUI_Toggle * = 0, DUI_Toggle * = 0,
    DUI_Toggle * = 0,          DUI_Toggle * = 0, DUI_Toggle * = 0, DUI_Toggle * = 0 );
    virtual void append_toggle( DUI_Toggle *,          DUI_Toggle * = 0, DUI_Toggle * = 0,
    DUI_Toggle * = 0,          DUI_Toggle * = 0, DUI_Toggle * = 0, DUI_Toggle * = 0,
    DUI_Toggle * = 0, DUI_Toggle * = 0, DUI_Toggle * = 0 );
    virtual void select( const DUI_Component * );
    virtual void select( const char * );
    virtual void append( const char *label, const char *flags );
    virtual void append( DUI_Component *, const char *flags );
    virtual void insert( int, const char *label, const char *flags );
    virtual void insert( int, DUI_Component *, const char *flags );
    virtual DUI_Component *remove( int );
    virtual DUI_Component *remove( int , STRING &flags );
    virtual const char * flags( int ) const;
    virtual void flags( int, const char * );
    virtual void append( const char *);
    virtual void append( DUI_Component * , DUI_Component * = 0,          DUI_Component
    * = 0, DUI_Component *
    Flag_Selection(2) Gatec Manual Flag_Selection(2)
    = 0,          DUI_Component * = 0, DUI_Component * = 0,          DUI_Component * = 0,
    DUI_Component * = 0,          DUI_Component * = 0, DUI_Component * = 0 );
    virtual void insert( int, const char *);
    virtual void insert( int, DUI_Component *);
protected:
friend class Session;
    void receive();
}
```

DESCRIPTION

This class is used by the Quote\_Abstract\_Form for the selection that displays the quotes. It is used to update the flags (on that form) that display information about the quote whenever a a quote is selected. Basically it is a DUI\_Selection that can have

toggles attached to it that get updated whenever the selection status changes. The values for each toggle are kept in a string of 0's and 1's which is associated with an entry in the selection.

## MEMBER FUNCTIONS

`t1, Flag_Selection::Flag_Selection( const char *name, DUI_Toggle * t1, DUI_Toggle * t2, DUI_Toggle * t3, DUI_Toggle * t4, DUI_Toggle * t5, DUI_Toggle * t6, DUI_Toggle * t7, DUI_Toggle * t8, DUI_Toggle * t9, DUI_Toggle * t10 )` Description: Constructor for Flag\_Selection accepting a name and up to ten toggles. returns: void

`Flag_Selection::~~Flag_Selection()`  
Description: Destructor for Flag\_Selection. returns: void

`void Flag_Selection::append_toggle( DUI_Toggle * t1, DUI_Toggle * t2, DUI_Toggle * t3, DUI_Toggle * t4, DUI_Toggle * t5, DUI_Toggle * t6, DUI_Toggle * t7, DUI_Toggle * t8, DUI_Toggle * t9, DUI_Toggle * t10 )`  
Description: Append up to ten more toggles to this Flag\_Selection. returns: void

`Flag_Selection::append( DUI_Component *component, const char *label, const char *flags )`  
Description: Appends a component and its flags to this selection. The flags specify the toggle states for this component. returns: void

`Flag_Selection::append( const char *label, const char *flags )`  
Description: Appends a label to the selection with its flags. returns: void

`Flag_Selection::insert( int i, DUI_Component *c, const char *label, const char *flags )`  
Description: inserts a component into this selection with its flags. returns: void

`Flag_Selection::insert( int i, const char *label, const char *flags )`  
Description: Inserts a label into this selection with its flags. returns: void

`Flag_Selection::remove( int i )`  
Description: Removes a component and its flags from the selection. returns: DUI\_Component \*, the removed component.

`Flag_Selection::remove( int i, STRING &flags )`  
Description: Removes a component and its flags but sets the flags argument to the flags that were removed. returns: DUI\_Component \*, the removed component.

Flag\_Selection::flags( int i )

Description: Accessor function. returns: char \*, the flags for the item indexed by i.

Flag\_Selection::flags( int i, const char \*flags )

Description: Sets the flags for item "i" in the selection. returns: void

void Flag\_Selection::append( DUI\_Component \*c1, DUI\_Component \*c2, DUI\_Component \*c3, DUI\_Component \*c4, DUI\_Component \*c5, DUI\_Component \*c6, DUI\_Component \*c7, DUI\_Component \*c8, DUI\_Component \*c9, DUI\_Component \*c10 ) Description: Overloads the DUI\_Selection(which see) append routine. returns: void

void Flag\_Selection::append( const char \*label )

Description: Overloads the DUI\_Selection(which see) routine. returns: void

void Flag\_Selection::insert( int i, DUI\_Component \*c )

Description: Overloads the DUI\_Selection(which see) routine. returns: void

void Flag\_Selection::insert( int i, const char \*label )

Description: Overloads the DUI\_Selection(which see) routine. returns: void

Flag\_Selection::receive()

Description: Change the DUI\_Toggles to match flags for selected component Flag values are a string of chars, 0 = ON, 1 = OFF, T = TOGGLE, else = same . for example, (using 0-indexing): "0110T" means flag 0 and 4 OFF, 1 and 3 ON, 2 unchanged, 5 TOGGLE returns: void

## FILES

Flag\_Selection.C Flag\_Selection.H

#### 2.2.4.4 Message\_Form

##### NAME

Message\_Form - Form for displaying the message viewing screen.

##### SYNOPSIS

```
#include "Message_Form.H"
```

```
class Message_Form : public DUI_Form { public:
    enum { Error_Message = 0, Unread, Needs_Action, Internal_Note, Read, Sent, Same };
    static Message_Form *instance( const char *new_rfq_number, const char
    *new_line_item, int new_message_category = Same );
    static Message_Form *instance();
    static void count_by_rfq( const char *rfq_num, const char *line_item, STRING
    &counts );
    static char message_priority( const char *rfq_num, const char *line_item );
    ~Message_Form();
    void change_message_category();
    void compose_message();
    void needs_action();
    void action_complete();
    void delete_note();
    void reply();
    void load_message_block();
    void view_rfq();
    void view_message();
    void quit_messages();
    void display_messages( ostream & );
protected:
    Message_Form();
protected:
friend class Compose_Message_Form;
    void load_messages();
Gatec(2) Last change: Tue Jan 4 16:20:03 1994 1
```

```
Message_Form(2) Gatec Manual Message_Form(2)
private:
    static Message_Form *instance_;
    int category;
    int stealth;
    List_of(STRING) document_ids;
    List_of(STRING) froms;
    List_of(STRING) from_cages;
    List_of(STRING) tos;
    List_of(STRING) dates;
    List_of(STRING) subjects;
    List_of(STRING) ref_nums;
```



```

int last_message_index;
void change_category( int new_message_category );
void setup( const char *new_rfq_number, const char *new_line_item, int
new_message_category );
void add_message( const char *to, const char *from, const char *from_cage_code,
const char *date, const char *subject, const char *reference_num, const char
*document_id );
static int count_messages( int cat, const char *rfq_number, const char *line_item );
void count_all_messages();
void load_data();
void save_data();
void update_category( int new_category, int use_last_message = 0 );
DUI_Label **category_count;
DUI_Command **category_command;
DUI_Command **category_actions;
DUI_Label *category_label;
DUI_Label *category_text;
DUI_Field *rfq_number;
DUI_Field *line_item;
DUI_Selection *message_abstract;
DUI_Text *message_body;
DUI_Command *compose_cmd;
/*DUI_Command *view_message_cmd;*/ DUI_Command *needs_action_cmd;
DUI_Command *action_complete_cmd;
DUI_Command *delete_note_cmd;
DUI_Command *reply_cmd;
DUI_Command *load_more_cmd;
}

```

## DESCRIPTION

This form implements a simple mail viewer for messages across 6 categories:

Error, Unread, Needs Action, Note, Read, and Sent

It contains 6 DUI\_Commands to switch between these six categories as well as a DUI\_Selection which shows the list of messages in the current category, and a DUI\_Text that shows the current selected message in that category.

It accesses the following tables(see NARQ):

Document, Message, MessageTo, MessageFrom, MessageReference, MessageTextBody, SolicitationLineItemError, Text

There is only one instance of a message screen in an application at one time.

## MEMBER FUNCTIONS

Message\_Form::instance( const char \*new\_rfq\_number, const char \*new\_line\_item, int new\_category ) Description: Public access to constructor, provided because there is only one instance. Calls setup with passed data to set up this instance. returns: Message\_Form \*, the instance.

Message\_Form::setup( const char \*new\_rfq\_number, const char \*new\_line\_item, int new\_category ) Description: Loads message\_abstract with data. returns: void

Message\_Form::load\_messages()  
Description: Loads messages into selection. returns: void

Message\_Form::add\_message( const char \*to, const char \*from, const char \*from\_cage\_code, const char \*date, const char \*subject, const char \*reference\_num, const char \*document\_id ) Description: Adds one line to the message\_abstract and saves data to data\_arrays. returns: void

Message\_Form::count\_all\_messages()  
Description: Counts the number of messages in each category. NOTE that these counts reflect the number of 864s, but the message\_abstract selection has 1 entry per text per 864 returns: void

Message\_Form::Message\_Form()  
Description: Constructor for Message\_Form. All DUI widgets used are created here. returns: void

Message\_Form::~Message\_Form()  
Description: Destructor for Message\_Form. Deletes commands and resets instance. returns: void

Message\_Form::change\_category( int new\_category )  
Description: Changes the form to reflect a new category. returns: void

use\_last\_message )  
Message\_Form::update\_category( int new\_category, int Description: moves message to new\_category if last\_message == 1, moves message currently in message\_text otherwise moves currently selected message removes it from abstract, changes counts. returns: void

Message\_Form::change\_message\_category()  
Description: Callback for category buttons. Changes category depending on which category was selected. returns: void

Message\_Form::quit\_messages()

Description: Quit view. Updates category if an unread message was read. returns: void

Message\_Form::reply()

Description: Sends the Compose\_Message\_Form(which see) with current message included. returns: void

Message\_Form::view\_rfq()

Description: Bring up the Review\_RFQ\_Form(which see) if rfq\_number and line\_item are not empty. returns: void

Message\_Form::compose\_message()

Description: Sends the Compose\_Message\_Form. returns: void

Message\_Form::needs\_action()

Description: Marks a messages as needing action. returns: void

Message\_Form::action\_complete()

Description: Marks a message as no longer needing action. returns: void

Message\_Form::delete\_note()

Description: Deletes a note. returns: void

Message\_Form::load\_message\_block()

Description: Loads another block of messages - NOT USED CURRENTLY. returns: void

Message\_Form::view\_message()

Description: Loads message text into message\_body. returns: void

Message\_Form::count\_messages( int cat, const char \*rfq\_number, const char \*line\_item ) Description: Returns the number of Messages in category 'cat' for rfq\_number- line\_item. returns: int, the number of messages.

Message\_Form::count\_by\_rfq( const char \*rfq\_num, const char \*line\_item, STRING &counts ) Description: Fills in counts with number of messages in each message category. Fills in 1st found with abbreviation for first. returns: void  
Message\_Form::message\_priority( const char \*rfq\_num, const char \*line\_item ) Description: Returns the highest priority message referencing rfq\_num & line\_item only checks for Error, Unread, or Needs\_Action messages. This is used by the Workload\_Form for displaying a flag beside an RFQ in its list if it has a priority message. returns: char, 'E', 'U', 'N' or '' denoting

priority.

Message\_Form::display\_messages( ostream &out )

Description: Displays each message to out stream in a simple  
ascii format. returns: void

## FILES

Message.C Message.H

## 2.2.4.5 Quote\_Abstract\_Form

### NAME

Quote\_Abstract\_Form - Form for displaying an abstract of quotes for an RFQ.

### SYNOPSIS

```
#include "Quote_Abstract_Form.H"
```

```
class Quote_Abstract_Form : public DUI_Form { public:
    static Quote_Abstract_Form *instance();
    static Quote_Abstract_Form *instance(    const char *new_rfq_number,    const char
    *new_line_item,    const char *new_req_number,    const char *new_fsc,    const char
    *new_fsc_suffix,    const char *new_priority,    const char *new_stock_number,    const char
    *new_estimated_price,    const char *new_quantity,    const char *new_unit,    const char
    *new_extended_price,    const char *new_sic,    const DUI_Text *new_item_description,    int
    new_amended,    const DUI_Text *new_price_history,    int category);
    static Quote_Abstract_Form *instance(    const char *new_rfq_number,    const char
    *new_line_item,    int category);
    ~Quote_Abstract_Form();
    void view_messages();
    void review_quote();
    void add_quote();
    void make_award();
    void confirm_make_award();
    void cancel_make_award();
    void hold_rfq();
    void redirect_rfq();
    void review_rfq();
    static void clear_data();
    void display_abstract( ostream & );
protected:
private:
    static Quote_Abstract_Form *instance_;
    int category_;
    void setup(    const char *new_rfq_number,    const char *new_line_item,    const char
    *new_req_number,    const char *new_fsc,    const char *new_fsc_suffix,    const char
    *new_priority,    const char *new_stock_number,    const char *new_estimated_price,    const char
    *new_quantity,    const char *new_unit,    const char *new_extended_price,    const char *new_sic,
    const DUI_Text *new_item_description,    int new_amended,    const DUI_Text
    *new_price_history,    int category);
    void load_quotes( const char *rfq_num, const char *line_item );
    int unread_messages_for_vendor(    const char *vendor_id,    const char
    *utn_number    );
    DUI_Field *rfq_number;
    DUI_Field *line_item;
    DUI_Field *requisition_number;
    DUI_Field *fsc;
```

```

DUI_Field *fsc_suffix;
DUI_Field *priority;
DUI_Field *message_count;
DUI_Toggle *amended;
    DUI_Field *stock_number;
DUI_Text *item_description;
DUI_Field *quantity;
DUI_Field *estimated_price;
DUI_Field *unit;
DUI_Field *extended_price;
DUI_Text *price_history;
    Flag_Selection *quote_selection;
DUI_Group *flag_group;
DUI_Toggle **quote_flags;
    DUI_Command *award_cmd;
DUI_Command *add_quote_cmd;
    DUI_Command *hold_rfq_cmd;
DUI_Command *redirect_rfq_cmd;
    DUI_Command *quit_cmd;
    Callback *confirm_make_award_callback;
    Callback *cancel_make_award_callback;
    List_of(STRING) *quote_ids;
#ifdef NODB    ReqForQuote *rfq;
#endif

```

```

STRING sic;
int num_low_quotes;
static float low_small_business_price_;
private:
friend class Award_Form;
    static float low_small_business_price() { return low_small_business_price_; }
}

```

#### DESCRIPTION

This form displays a list of the quotes received for a closed RFQ. The Quotes are displayed in a Flag\_Selection(which see) which has a list of toggles associated with it that are turned on and off depending on which quote is selected. This form is instantiated by the Workload\_Form under the closed category. It allows the RFQ to be awarded to one of the quotes listed. It also allows the RFQ to be held or redirected. The Forms that can be instantiated by this form are:

Message\_Form(which see), Review\_Quote\_Form(which see), Award\_Form(which see), Review\_RFQ\_Form(which see)

This form accesses the following tables:

Acquisition(which see), Document(which see), ReqForQuote(which see), ReqForQuoteLineItem(which see), LineItem(which see), Part(which see), Quote(which see),

QuoteLineItem(which see), QuoteTerms(which see), Variations(which see), RelatedPaperwork(which see), SolicitationLineItem(which see), SolicitationHistory(which see), MessageFrom(which see), Vendor(which see)

There is only one instance of the form in an application at one time.

## MEMBER FUNCTIONS

Quote\_Abstract\_Form \* Quote\_Abstract\_Form::instance\_ = 0;

float Quote\_Abstract\_Form::low\_small\_business\_price\_ = 0;

Quote\_Abstract\_Form \* Quote\_Abstract\_Form::instance()

Description: This function returns the instance\_pointer whatever the value is. returns:

Quote\_Abstract\_Form \*, the instance or 0.

Quote\_Abstract\_Form::instance( const char \*new\_rfq\_number, const char \*new\_line\_item, const char \*new\_req\_number, const char \*new\_fsc, const char \*new\_fsc\_suffix, const char \*new\_priority, const char \*new\_stock\_number, const char \*new\_estimated\_price, const char \*new\_quantity, const char \*new\_unit, const char \*new\_extended\_price, const char \*new\_sic, const DUI\_Text \*new\_item\_description, int new\_amended, const DUI\_Text \*new\_price\_history, int category ) Description:

Public access to constructor, provided because there is only one instance. Calls setup() to set up the instance. returns:

Quote\_Abstract\_Form \*, the instance.

Quote\_Abstract\_Form::instance( const char \*new\_rfq\_number, const char \*new\_line\_item, int category ) Description: Public access to constructor, provided because there is only one instance. returns:

Quote\_Abstract\_Form \*, the instance.

Quote\_Abstract\_Form::setup( const char \*new\_rfq\_number, const char \*new\_line\_item, const char \*new\_req\_number, const char \*new\_fsc, const char \*new\_fsc\_suffix, const char \*new\_priority, const char \*new\_stock\_number, const char \*new\_estimated\_price, const char \*new\_quantity, const char \*new\_unit, const char \*new\_extended\_price, const char \*new\_sic, const DUI\_Text \*new\_item\_description, int new\_amended, const DUI\_Text \*new\_price\_history, int category ) Description: Sets up this instance given the arguments. Calls load\_quotes() to fill the quote list. returns: void

Quote\_Abstract\_Form::~Quote\_Abstract\_Form()

Description: Destructor. Resets instance, removes all quotes and quote flags, deletes "rfq" table. returns: void

Quote\_Abstract\_Form::Quote\_Abstract\_Form()

Description: Constructor for Quote\_Abstract\_Form. Instantiates "rfq" table and all of the DUI\_Widgets required for this form. returns: void

\*lineItem)

Quote\_Abstract\_Form::load\_quotes( const char \*rfq\_num, const char Description: Loads the quote abstract with the quotes for this RFQ setting up the flags as well. returns:

void

Quote\_Abstract\_Form::view\_messages()

Description: Pop up the Message\_form with correct rfq and line\_item. returns: void

Quote\_Abstract\_Form::review\_quote()

Description: Review the selected quote. Instantiates the Review\_Quote\_Form (which see). returns: void

Quote\_Abstract\_Form::make\_award()

Description: Calls confirm\_make\_award() if user has selected the lowest quote, otherwise pops up a dialog warning the user. returns: void

Quote\_Abstract\_Form::confirm\_make\_award()

Description: Brings up the Award\_Form(which see). returns: void

Quote\_Abstract\_Form::cancel\_make\_award()

Description: Brings up the quote abstract again. returns: void

Quote\_Abstract\_Form::review\_rfq()

Description: Instantiates the Review\_RFQ\_Form(which see). returns: void

Quote\_Abstract\_Form::hold\_rfq()

Description: Instantiates the Review\_RFQ\_Form(which see) and calls its hold\_rfq(). returns: void

Quote\_Abstract\_Form::redirect\_rfq()

Description: Instantiates the Review\_RFQ\_Form(which see) and calls its redirect\_rfq(). returns: void

Quote\_Abstract\_Form::add\_quote()

Description: Instantiates the Review\_Quote\_Form(which see) which can act as a data entry screen for adding a quote from scratch. returns: void

Quote\_Abstract\_Form::clear\_data()

Description: Clears all the fields on this form. returns: void

Quote\_Abstract\_Form::display\_abstract( ostream &out )

Description: Print a simple ascii representation of the quote list, Award\_Form data if there is any, and the quote flags onto stream "out". returns: void

Quote\_Abstract\_Form::unread\_messages\_for\_vendor(

const char \*vendor\_id, const char \*utn\_number  
) Description: Checks to see if there are any unread messages



for the passed vendorid, and utnnumber. returns: int 1 if yes, 0 if no.

## FILES

Quote\_Abstract.C Quote\_Abstract.H

### 2.2.4.6 RFQ\_Category

#### NAME

RFQ\_Category - Class to handle the lists of RFQs in each workload category.

#### SYNOPSIS

```
#include "RFQ_Category.H"
```

```
class RFQ_Category { private:
static RFQ_Category *category_ptrs[ NUM_CATEGORIES ];
static int total_rfqs;
    DUI_Group      *rfq_group_;
    List_of(STRING) doc_ids_;
    DUI_Label      *count_label_;
    DUI_Command    *command_;
    short database_queried_;
    int  rfqs_in_category_;
    int  max_rfqs_shown_;
    int  category_;
public:
static void initialize( DUI_View *view, Callback *callback );
static RFQ_Category *instance( int i ) { return category_ptrs[i]; }
    void append_rfq( DUI_Component *rfq_label, const char *doc_id );
    void get_rfq( int i, DUI_Component *rfq_label, STRING &doc_id );
    int move_rfq( int i, int new_category );
    int move_rfq( const char *document_id, int new_category );
    int remove_rfq( const char *document_id );
    void remove_rfq( int i );
    void load_rfqs( int show_more_rfqs = 0 );
    DUI_Label *count_label() { return count_label_; }
    DUI_Command *command() { return command_; }
    DUI_Group *rfq_group() { return rfq_group_; }
    short database_queried() { return database_queried_; }
    void database_queried( int l ) { database_queried_ = l; }
}
    const char * name() { return category_name[ category_ ]; }
    const char * description() { return category_desc[ category_ ]; }
    const char * db_code() { return category_db[ category_ ]; }
    int rfqs_shown() { return rfq_group_->component_count(); }
    void update_rfq( const char *rfq, const char *line_item, const char *fsc, const char
*item_description );
    void update_rfq( const char *rfq, const char *line_item, char message_priority );
private:
    RFQ_Category( int cat, DUI_Command *command );
    int count_rfqs();
    void query_rfqs();
    void change_count_label();
```

```

private:
    Document *document_;
    Acquisition *acquisition_;
    ReqForQuote *rfq_;
    FetchedRows *doc_rows_;
    ComplexQuery *doc_query_;
}

```

## DESCRIPTION

This class is used to retrieve and control the lists of RFQs in each of the buyer workload categories:

Unissued, Unissued Held, Open, Closed, Closed Held, Overdue, Awarded

There is an instance of this class for each of these categories.

It queries the database as little as possible to increase speed, but this can mean that what is contained in this class does not necessarily reflect the state of the data base at every moment. It queries the following tables:

Acquisition(which see), Document(which see),  
ReqForQuoteLineItem(which see), ReqForQuote(which see)

## MEMBER FUNCTIONS

RFQ\_Category::initialize( DUI\_View \*view, Callback \*callback )

Description: Public access to set up all categories, it creates each Category, counts RFQs, and changes count labels. returns: void

RFQ\_Category::RFQ\_Category( int cat, DUI\_Command \*command )

Description: Constructor for RFQ\_Category class. Creates tables and initializes controls. returns: void

RFQ\_Category::append\_rfq( DUI\_Component \*rfq\_label, const char  
Description: Adds an RFQ to the current category. returns: void

RFQ\_Category::get\_rfq( int i, DUI\_Component \*rfq\_label, STRING  
Description: Returns an RFQ from this category by setting the last two arguments. returns: void

)  
RFQ\_Category::move\_rfq( const char \*document\_id, int new\_category  
Description: Moves RFQ with document\_id from this category to new\_category. returns: the return value of move\_rfq(int, int) (1 if successful, 0 otherwise).

RFQ\_Category::move\_rfq( int i, int new\_category )

Description: Moves RFQ i from this category to new\_category and loads the next RFQ in this category. returns: int, 1 when

successful, 0 otherwise.

RFQ\_Category::remove\_rfq( const char \*document\_id )

Description: Removes RFQ with document\_id from this category. returns: 1 if successful, 0 otherwise.

RFQ\_Category::remove\_rfq( int i )

Description: Removes RFQ i from this category and loads next RFQ in this category. returns: void

RFQ\_Category::count\_rfqs()

Description: Counts RFQs in current category. returns: int, count.

RFQ\_Category::change\_count\_label()

Description: Changes the label for this RFQ\_Category to reflect count and percentage. returns: void

RFQ\_Category::load\_rfqs( int show\_more\_rfqs )

Description: Loads rfqs from database - shows "show\_more\_rfqs" more rfqs than before. returns: void

RFQ\_Category::query\_rfqs()

Description: Create query for RFQs. returns: void

RFQ\_Category::update\_rfq( const char \*rfq, const char \*line\_item, const char \*fsc, const char \*item\_description ) Description: Updates RFQ line when user changes item\_description or fsc. returns: void

RFQ\_Category::update\_rfq( const char \*rfq, const char \*line\_item, char message\_priority ) Description: Updates RFQ line when user changes message\_priority. returns: void

## FILES

RFQ\_Category.C RFQ\_Category.H

### 2.2.4.7 Review\_Quote\_Form

NAME

Review\_Quote\_Form - Form for reviewing or adding a quote.

SYNOPSIS

```
#include "Review_Quote_Form.H"
```

```
class Review_Quote_Form : public DUI_Form { public:
    static Review_Quote_Form *instance( const char *rfq, const char *line, const char *quote_id,
const char *flag_values, const DUI_Text *item_desc, int category );
    static Review_Quote_Form *instance( const char *rfq, const char *line, const char
*stock_num, const char *est_price, const char *fsc_value, const char *sic_value, const
DUI_Text *item_desc );
    ~Review_Quote_Form();
    void load_data( const char *rfq, const char *line, const char *quote_id, const char
*flag_values, const DUI_Text *item_desc );
    void view_messages();
    void compose_message();
    void vendor_info();
    void make_award();
    void commit_add();
    static void clear_data();
protected:
    Review_Quote_Form();
private:
    static Review_Quote_Form *instance_;
    void review_quote( const char *rfq, const char *line, const char *quote_id, const char
*flag_values, const DUI_Text *item_desc, int category );
    void add_quote( const char *rfq, const char *line, const char *stock_num, const char
*est_price, const char *fsc_value, const char *sic_value, const DUI_Text *item_desc );
    void change_read_only( boolean ro );
    DUI_Field *rfq_number;
    DUI_Field *line_item;
    DUI_Field *stock_number;
    DUI_Field *estimated_price;
    DUI_Field *fsc;
    DUI_Field *sic;
    DUI_Text *item_description;
    DUI_Field *vendor_cage_code;
    DUI_Field *vendor_name;
    DUI_Field *quote_effective_date;
    DUI_Field *quote_expires_date;
    DUI_Field *unit_price;
    DUI_Field *quantity;
    DUI_Field *unit;
    DUI_Field *extended_price;
    DUI_Field *delivery_date;
```

```

DUI_Field *discount_percent;
DUI_Field *discount_due_days;
DUI_Field *discount_net_due_days;
DUI_Field *variation;
DUI_Field *fob;
DUI_Text *quote_description;
DUI_Field *vendor_note;
DUI_Field *flags;
DUI_Field *requirements_contract;
DUI_Field *fss_contract;
DUI_Field *contract_expiration_date;
    DUI_Toggle *small_business;
DUI_Group *changing_group;
    DUI_Command *review_cmds;
DUI_Command *add_cmds;
DUI_Command *quit_cmd;
DUI_Command *cancel_add_cmd;
DUI_Command *award_cmd;
#ifdef NODB    Quote *quote;
    QuoteLineItem *quote_li;
    Part *part;
    ReqForQuoteLineItem *rfq_li;
    ReqForQuote *rfq;
    FreeOnBoard *fob_table;
    QuoteTerms *qt;
#endif }

```

## DESCRIPTION

This class is used to review the contents of a quote. It queries the following tables:

Quote(which see), QuoteLineItem(which see), Part(which see), ReqForQuoteLineItem(which see), ReqForQuote(which see), Message(which see), RelatedPaperwork(which see), FreeOnBoard(which see), QuoteTerms(which see), Document(which see)

There is only one instance of this form in an application at one time.

## MEMBER FUNCTIONS

```

Review_Quote_Form *Review_Quote_Form::instance_ = 0;
Review_Quote_Form * Review_Quote_Form::instance( const char *rfq, const char *line,
const char *quote_id, const char *flag_values, const DUI_Text *item_desc, int category )
Description: Public access to constructor, provided because there is only one instance. This
instance used for reviewing a quote. returns:
Review_Quote_Form *, the instance.

```

`Review_Quote_Form::instance( const char *rfq, const char *line, const char *stock_num, const char *est_price, const char *fsc_value, const char *sic_value, const DUI_Text *item_desc )` Description: This instance used for adding a quote. returns: `Review_Quote_Form *`, the instance.

`Review_Quote_Form::Review_Quote_Form()`  
Description: Constructor for `Review_Quote_Form`. Instantiates all the `DUI_Widgets` used by this form. returns: `void`

`Review_Quote_Form::~~Review_Quote_Form()`  
Description: Destructor for `Review_Quote_Form`. Deletes flags and tables. returns: `void`

`Review_Quote_Form::change_read_only( boolean ro )`  
Description: Changes appropriate field's `read_only` status. returns: `void`

`Review_Quote_Form::review_quote( const char *rfq, const char *line, const char *quote_id, const char *flag_values, const DUI_Text *item_desc, int category )` Description: Sets up form for reviewing quotes. returns: `void`

`Review_Quote_Form::add_quote( const char *rfq, const char *line, const char *stock_num, const char *est_price, const char *fsc_value, const char *sic_value, const DUI_Text *item_desc )` Description: Sets up form for adding quotes. returns: `void`

`Review_Quote_Form::clear_data()`  
Description: Clears all components. returns: `void`

`load_item_desc( QuoteLineItem *quote_li, DUI_Text *item_desc )`  
Description: Loads the item description text ( this will change as the database changes ). returns: `void`

`Review_Quote_Form::load_data( const char *rfq_num, const char *line, const char *quote_id, const char *flag_values, const DUI_Text *item_desc )` Description: Loads quote data from database. returns: `void`

`Review_Quote_Form::view_messages()`  
Description: Instantiates and displays the `Message_Form`(which see). returns: `void`

`Review_Quote_Form::compose_message()`  
Description: Instantiates and displays the `Compose_Message_Form`(which see) with quote reference number passed in. returns: `void`

`Review_Quote_Form::vendor_info()`  
Description: Goes to Vendor performance screen. Not implemented. returns: `void`

`Review_Quote_Form::make_award()`  
Description: Makes the award. Instantiates the `Quote_Abstract_Form`(which see) and calls its `make_award()`. returns: `void`

`Review_Quote_Form::commit_add()`

Description: Adds this Quote into the database. returns: void

FILES

Review\_Quote.C Review\_Quote.H



#### 2.2.4.8 Review\_RFQ\_Form

##### NAME

Review\_RFQ\_Form - form for reviewing an RFQ.

##### SYNOPSIS

```
#include "Review_RFQ_Form.H"
```

```
class Review_RFQ_Form: public DUI_Form { public:
    static Review_RFQ_Form *instance( const char *rfq, const char *line,      int category =
SAME_CATEGORY );
    ~Review_RFQ_Form();
    void view_messages();
    void more_info();
    void issue_rfq();
    void add_quote();
    void save_and_hold_rfq();
    void save_and_redirect_rfq();
    void save_and_view_abstract();
    void amend_rfq();
    void cancel_rfq();
    void quit_review();
    void view_abstract();
    void hold_rfq();
    void redirect_rfq();
    void commit_redirect();
    void commit_hold();
    void cancel_hold();
    void save_and_quit();
    void commit_quit();
    void cancel_quit();
    void confirm_cancel_rfq();
    void confirm_amend_rfq();
    void cancel_amend_rfq();
    void confirm_upload();
    static void clear_data();
protected:
    Review_RFQ_Form();
private:
    static Review_RFQ_Form *instance_;
    int save_ok;
    int acq_save_ok;
    int category_;
    DUI_Label *category_label;
    DUI_Label *category_text;
    DUI_Group *info_holder;
    DUI_Group **info_group;
    char *upload_filename;
```

```

List_of(String) vendor_id_list;
int current_info;
void set_values( const char *rfq, const char *line, int category );
void change_category();
void load_data( const char *rfq_num, const char *rfq_line_item );
void save_data();
int save_data_no_commit();
void change_info();
void make_read_write();
void make_read_only();
int verify_addressees();
int issue_840( const char *send_status );
void upload_to_standard_system();
void print_rfq();
#ifdef CDF void generate_840( String & );
#endif

```

```

DUI_Field *rfq_number;
DUI_Field *line_item;
DUI_Field *fsc;
DUI_Field *fsc_suffix;
DUI_Text *fsc_suffix_description;
DUI_Field *sic;
DUI_Field *requisition_number;
DUI_Field *stock_number;
DUI_Text *item_description;
DUI_Field *quantity;
DUI_Field *unit;
DUI_Field *required_response_date;
DUI_Field *required_delivery_date;
DUI_Field *ship_to_zip;
DUI_Field *manufacturer;
DUI_Field *part_number;
DUI_Field *message_count;
DUI_Toggle *amended;
DUI_Toggle *upload_changes;
DUI_Group *component_group;
DUI_Label *hold_info;
DUI_Text *addressees;
DUI_Text *additional_clauses;
DUI_Field *paperwork_required;
DUI_Selection *paperwork_received;
DUI_Field *rfq_date;
DUI_Field *buyer_code;
DUI_Field *priority;
DUI_Field *extended_price;
DUI_Field *estimated_price;
DUI_Field *fund_code;
DUI_Field *sran;
DUI_Field *project_code;

```

```

DUI_Field *bn_ss;
DUI_Field *defense_priority_rating;
DUI_Field *requestor_name;
DUI_Field *requestor_phone;
DUI_Text *requestor_source;
DUI_Field *project_title;
DUI_Text *procurement_history;
    DUI_Group    *hold_group;
DUI_Group    *hold_shared_group;
DUI_Field    *hold_rfq_number;
DUI_Field    *hold_line_item;
DUI_Field    *hold_expiration;
DUI_Selection *hold_reason;
Callback     *hold_callback;
Callback     *commit_hold_callback;
    DUI_Group    *redirect_group;
DUI_Group    *redirect_shared_group;
DUI_Field    *redirect_rfq_number;
DUI_Field    *redirect_line_item;
DUI_Toggle   *print_new_abstract;
DUI_Toggle   *print_rfq_info;
DUI_Selection *redirect_reason;
Callback     *redirect_callback;
Callback     *commit_redirect_callback;
    Callback     *save_and_quit_callback;
Callback     *commit_quit_callback;
Callback     *cancel_quit_callback;
Callback     *cancel_callback;
Callback     *confirm_upload_callback;
DUI_Command *    print_rfq_cmd;
DUI_Command *    issue_rfq_cmd;
DUI_Command *    redirect_rfq_cmd;
DUI_Command *    hold_rfq_cmd;
DUI_Command *    amend_rfq_cmd;
DUI_Command *    cancel_rfq_cmd;
DUI_Command * confirm_amend_rfq_cmd;
DUI_Command * cancel_amend_rfq_cmd;
DUI_Command *    add_quote_cmd;
DUI_Command *    view_abstract_cmd;
DUI_Command *    quit_cmd;
    DUI_Command * category_cmds;
DUI_Command * other_cmds;
#ifdef NODB  ReqForQuote *rfq;
ReqForQuoteLineItem *rfq_li;
Part *part;
Part *mf_part;
Document *doc;
Acquisition *acq;
Clause *clause;
DocumentAddressee *doc_addr;

```

```

ShippingDocPackage *shipdoc;
GSDefaults *gsd;
ISADefaults *isad;
List_of(STRING) shipdoc_keys;
List_of(STRING) clause_keys;
SolicitationLineItem *sol_li;
#endif }

```

## DESCRIPTION

This form is used to review an RFQ that has been selected from the Workload\_Form. It is also instantiated from other forms to review the rfq they are dealing with or to redirect or hold the rfq they are dealing with. There are two screens of information for an RFQ. This class handles both. They are toggled using the "More Info" command.

It accesses the following tables:

ReqForQuote(which see), ReqForQuoteLineItem(which see), Part(which see), Document(which see), DocumentSent(which see), Acquisition(which see), DocumentAddressee(which see), Clause(which see), ShippingDocPackage(which see), GSDefaults(which see), ISADefaults(which see), Item(which see), MeasurementData(which see), PriorityGroup(which see), RelatedPaperwork(which see), SolicitationHistory(which see), Unit(which see), SolicitationLineItem(which see), Message(which see), Vendor(which see), UserManagerDefaults(which see)

There is only one instance of this form in an application at one time.

## MEMBER FUNCTIONS

```
Review_RFQ_Form * Review_RFQ_Form::instance_ = 0;
```

```
Review_RFQ_Form * Review_RFQ_Form::instance( const char *rfq, const char *line, int
cat ) Description:
```

Public access to constructor, provided because there is only one instance. Calls set\_values() to set up the instance. returns: Review\_RFQ\_Form \*, the instance.

```
int cat )
```

Review\_RFQ\_Form::set\_values( const char \*rfq, const char \*line, Description: Sets up instance for this "rfq". It switches the to 1st rfq screen if it is not already there. returns: void

```
Review_RFQ_Form::Review_RFQ_Form()
```

Description: Constructor for Review RFQ form, public access through instance() fxn. Instantiates all the DUI\_Widgets used by this form. returns: void

```
Review_RFQ_Form::change_category( )
```

Description: Modifies category labels and commands. Also,

adds hold information if needed. returns: void

Review\_RFQ\_Form::make\_read\_only()

Description: Makes all values read\_only (for Open and Closed RFQs). returns: void

Review\_RFQ\_Form::make\_read\_write()

Description: Make values specified in FRD 3.2.2 read\_write (Unissued and Revised RFQs). returns: void

Review\_RFQ\_Form::clear\_data()

Description: Clears value from fields on screen. returns: void

)

load\_item\_desc( ReqForQuoteLineItem \*rfq\_li, DUI\_Text \*item\_desc Description: Loads the item description text (this will change as the database changes). returns: void \*rfq\_line\_item )

Review\_RFQ\_Form::load\_data( const char \*rfq\_num, const char Description: Loads data from database. returns: void

Review\_RFQ\_Form::change\_info()

Description: Changes what information is on the screen. Toggles screens. returns: void

Review\_RFQ\_Form::view\_messages()

Description: Displays messages related to this RFQ. Instantiates the Message\_Form(which see). returns: void

Review\_RFQ\_Form::more\_info()

Description: Shows next screen of information for this RFQ. returns: void

Review\_RFQ\_Form::amend\_rfq()

Description: Amend RFQ - brings up the Review RFQ Form in read\_write mode with amend\_cmds. returns: void

Review\_RFQ\_Form::confirm\_amend\_rfq()

Description: Issues an amended RFQ, returns to current category. returns: void

Review\_RFQ\_Form::cancel\_amend\_rfq()

Description: Cancels amend, returns to current category. returns: void

Review\_RFQ\_Form::cancel\_rfq()

Description: Prompts the user to make sure they want to cancel this RFQ. returns: void

Review\_RFQ\_Form::confirm\_cancel\_rfq()

Description: Issues a canceled RFQ, returns to current category.  
returns: void

Review\_RFQ\_Form::issue\_rfq()

Description: Issues the RFQ. returns: void

int Review\_RFQ\_Form::verify\_addressees()

Description: Verify that cage\_codes exist in database. returns: int,  
1 if success 0 if failure.

Review\_RFQ\_Form::issue\_840( const char \*send\_status )

Description: Issues the RFQ (X12 840), issue as amended RFQ if  
it is not from the Unissued categor. returns: 1 if successful, 0  
otherwise.

)

save\_item\_desc( ReqForQuoteLineItem \*rfq\_li, DUI\_Text  
\*item\_desc Description: Saves item description to database (will  
change as database changes). returns: void

Review\_RFQ\_Form::save\_data()

Description: Saves field values to database and commits database  
if success. returns: void

Review\_RFQ\_Form::save\_data\_no\_commit()

Description: Saves data to database but doesn't commit. returns:  
int, 1 if save succeeded, 0 otherwise.

Review\_RFQ\_Form::generate\_840( STRING &retval )

Description: Queries tables required by \_840DBtoCDF()  
but have not been queried yet and calls \_840DBtoCDF(). returns:  
void

Review\_RFQ\_Form::confirm\_upload()

Description: Pops current entry from the bcasitem queue, adds  
new cdf. returns: void

Review\_RFQ\_Form::save\_and\_redirect\_rfq()

Description: Pops up a dialog prompting for the reason for  
redirect. returns: void

Review\_RFQ\_Form::redirect\_rfq()

Description: Pops up the redirect dialog. returns:  
void

Review\_RFQ\_Form::commit\_redirect()

Description: Actually redirect the RFQ. returns: void

Review\_RFQ\_Form::save\_and\_hold\_rfq()

Description: Pops up a dialog prompting for the reason for hold

and hold expiration date by calling hold\_rfq()  
after saving data. returns: void

Review\_RFQ\_Form::hold\_rfq()  
Description: Actually pops up the dialog. returns:  
void

Review\_RFQ\_Form::commit\_hold()  
Description: Put RFQ on hold. returns: void

Review\_RFQ\_Form::add\_quote()  
Description: Manually add a quote - LEAD BUYER ONLY  
(though there is no check). returns: void

Review\_RFQ\_Form::save\_and\_view\_abstract()  
Description: view abstract of quotes for this RFQ NOTE:  
since this form never displays this button, the queries for finding  
this rfq\_information should probably be moved into  
Quote\_Abstract\_Form However, this works for now, it's just not as  
clean. returns: void

Review\_RFQ\_Form::view\_abstract()  
Description: Instantiates the Quote\_Abstract\_Form(which see).  
returns: void

Review\_RFQ\_Form::quit\_review()  
Description: Checks for changes before quitting - if changes,  
prompts the user for whether or not to save. returns: void

Review\_RFQ\_Form::save\_and\_quit()  
Description: Saves changes and quits. returns: void

Review\_RFQ\_Form::commit\_quit()  
Description: Quit review -reloads Workload\_Form(which see).  
returns: void

Review\_RFQ\_Form::cancel\_quit()  
Description: Cancels quit. returns: void

Review\_RFQ\_Form::~~Review\_RFQ\_Form()  
Description: Destructor for Review\_RFQ\_Form, sets instance\_  
= 0. returns: void

Review\_RFQ\_Form::print\_rfq()  
Description: Sends the RFQ and Quote Abstract to the default  
printer. returns: void

## FILES

Review\_RFQ.C Review\_RFQ.H

### 2.2.4.9 Vendor\_Performance\_Data

#### NAME

Vendor\_Performance\_Form - form for displaying information about a vendor.

#### SYNOPSIS

```
#include "Vendor_Performance_Form.H"
```

```
class Vendor_Performance_Form : public DUI_Form { public:  
    static Vendor_Performance_Form *instance( );  
    ~Vendor_Performance_Form();  
    void load_data( );  
protected:  
    Vendor_Performance_Form();  
private:  
    static Vendor_Performance_Form *instance_;  
}
```

#### DESCRIPTION

This form is not implented yet.

#### MEMBER FUNCTIONS

#### FILES

Vendor\_Performance.C Vendor\_Performance.H



### 2.2.4.10 Workload\_Form

#### NAME

Workload\_Form - Form for displaying a list of rfqs by category.

#### SYNOPSIS

```
#include "Workload_Form.H"
```

```
class Workload_Form : public DUI_Form { public:
    static Workload_Form *instance();
    ~Workload_Form();
    void change_category();
    void view_messages();
    void view_unread_messages();
    void view_errors();
    void review_rfq();
    void find_rfq();
    void review_quotes();
    void select_next_rfq();
    void view_next_rfq();
    void quit_workload();
    void select( int cat, int n );
    int selection( STRING &rfq, STRING &line );
protected:
    Workload_Form();
private:
    static Workload_Form *instance_;
    int current_category;
    DUI_Group *unread_group_;
    void change_category( int category );
    void review_found_rfq();
    void review_found_award();
    void load_more_rfqs();
    DUI_Command *find_rfq_cmd;
    DUI_Command *review_rfq_cmd;
    DUI_Command *review_quotes_cmd;
    DUI_Command *other_cmds;
    DUI_Command *rfq_cmds;
    DUI_Label *category_label;
    DUI_Label *category_text;
    DUI_Selection *rfq_selection;
    DUI_Group *find_rfq_group;
    DUI_Field *rfq_number;
    DUI_Field *line_item;
    Callback *find_rfq_callback;
    DUI_Field *award_number;
    Callback *find_award_callback;
```

}  
DESCRIPTION

This form is the first form in the GATEC application. It displays the buyer's workload by listing the rfqs in each of the following categories:

Unissued, Unissued Held, Open, Closed, Closed Held, Overdue, Awarded

The buyer can switch between these categories by selecting commands embedded in the form. It uses RFQ\_Category(which see) to keep track of the categories. It accesses the following tables:

Award(which see), Acquisition(which see), Document(which see), ReqForQuote(which see), ReqForQuoteLineItem(which see), LineItem(which see), Message(which see), MessageReference(which see)

There is only one instance of this form in an application at one time.

#### MEMBER FUNCTIONS

Workload\_Form \*Workload\_Form::instance\_ = 0;

Workload\_Form \* Workload\_Form::instance() Description: Public access to constructor, provided because there is only one instance. returns: Workload\_Form \*, the instance.

Workload\_Form::~~Workload\_Form()

Description: Destructor. Resets instance\_. returns: void

Workload\_Form::Workload\_Form()

Description: Constructor for Workload\_Form. Instantiates all the DUI\_Widgets used in this form. returns: void

Workload\_Form::change\_category()

Description: A change\_category command was chosen. Calls change\_category(int). returns: void

Workload\_Form::change\_category( int cat )

Description: Change Workload\_Form to display category "cat", using RFQ\_Category(which see). returns: void

Workload\_Form::review\_rfq()

Description: Review the selected RFQ or all if none are selected. returns: void

Workload\_Form::review\_quotes()

Description: Goes directly to the Quote\_Abstract\_Form. returns:

void

Workload\_Form::find\_rfq()

Description: Pops up a dialog asking for RFQ number and line item to find. returns: void

Workload\_Form::review\_found\_rfq()

Description: Sends Review\_RFQ\_Form with found rfq. returns: void

Workload\_Form::review\_found\_award()

Description: Sends Quote\_Abstract\_Form with found award. returns: void

Workload\_Form::view\_messages()

Description: View messages (related to selected RFQ). returns: void

Workload\_Form::load\_more\_rfqs()

Description: Loads more RFQs to screen. returns: void

Workload\_Form::view\_unread\_messages()

Description: View unread messages (not related to any RFQ). returns: void

Workload\_Form::view\_errors()

Description: View error messages. returns: void

Workload\_Form::selection( STRING &rfq, STRING &line )

Description: Sets rfq and line arguments to current selection. returns: int, current category.

Workload\_Form::select( int cat, int n )

Description: If current\_category = cat, selects n'th RFQ. returns: void

Workload\_Form::select\_next\_rfq()

Description: Selects the next RFQ in the Workload list. returns: void

Workload\_Form::view\_next\_rfq()

Description: Displays the selected RFQ or Quotes depending on category. returns: void

Workload\_Form::quit\_workload()

Description: Forces a quit of the application. returns: void

## FILES

Workload.C Workload.H

---

## 2.3 Lead Buyer Application

---

Lead\_buyer.dui is an application that fulfills the user interface requirements for the lead buyer functions of the GATEC project. It allows the user to view statistics on the current status of the GATEC database. It does this by providing a way to query the GATEC system for procurement activities based on the following criteria:

RFQ Number, Stock Number, Stock Class, SRAN, Review Status, RFQ Date, BSP

The user can then view statistics and award history information on the list of activities that match the query criteria. In addition, the application allows the user to change the buyer associated with the matching list. In this way the user can adjust the workload assigned to each buyer.

It uses DUI(1) for its user interface and interacts with a data base through the NARQ(see NARQ) and NORA(see NORA) libraries. It is written in C++. To get a user perspective on the lead\_buyer.dui application see *Lead Buyer User's Guide* [REF000].

The following sections give a technical overview of the lead\_buyer.dui application.

---

### 2.3.1 Class Hierarchy

---

The lead\_buyer.dui application has the following class hierarchy, indentation denotes derivation:

(DUI\_Form) defined in DUI(1)  
Change\_RFQs\_Form  
List\_RFQs\_Form  
Price\_History\_Form  
Price\_Performance\_Form  
Select\_RFQs\_Form  
Statistics\_Form  
Summarized\_RFQ  
RFQ\_Summary  
Range\_List  
Sort\_Order  
String

The derivatives of DUI\_Form are all interface classes describing the forms used in the application. RFQ\_Summary(3) is the class that deals with the database. It does all the querying and generates a list of Summarized\_RFQ(3)'s which simply hold the information for any one one procurement activity. Range\_List(3) is a class for parsing strings containing lists of values or ranges of values(the user is allowed to enter ranges when specifying the values for the selection criteria). Sort\_Order(3) is a class for holding the order in which the Summarized\_RFQ's are to sorted. String is a generic string class.

See the individual documentation on these classes for more details.

---

### 2.3.2 Programming Hints

---

The documentation for the individual form classes should be consulted for the specific function of the lead\_buyer.oui application that they fulfill. Also the DUI and NARQ and NORA man pages should be consulted because this will clarify a lot of the code found in the form classes and RFQ\_Summary(3).

RFQ\_Summary is the class responsible for all the database querying functionality and for generating a list of matching procurement activities. This is the class to look in if there are problems with the records being queried. This class is passed around to the other forms for them to display or operate on.

Look in Summarized\_RFQ(3) for the actual data that is retrieved from the data base for each matching record.

The String class is identical to the DUI(1) string class but minus the communicable object stuff.

---

### 2.3.3 Lead Buyer Source Tree

---

The source for lead\_buyer.oui is kept under the DUI(1) source tree in:

`$CVSROOT/oui/applications/lead_buyer`

It depends on the NARQ and NORA libraries being in:  
`$CVSROOT/narqdb/lib`

These must be made before the lead\_buyer.oui application can be made. To make the lead\_buyer.oui application, cd to its source directory and type:

xmkmf; make depend all

The resulting "lead\_buyer.oui" file will be installed in:  
\$CVSROOT/oui/bin

---

### 2.3.4      Lead Buyer Form Classes

---

The lead\_buyer.oui forms are comprised of the following classes:

Change\_RFQs\_Form  
List\_RFQs\_Form  
Price\_History\_Form  
Price\_Performance\_Form  
RFQ\_Summary  
Range\_List  
Select\_RFQs\_Form  
Sort\_Order  
Statistics\_Form  
String  
Summarized\_RFQ

### 2.3.4.1 Change\_RFQs\_Form

NAME

Change\_RFQs\_Form - Defines the Screen that allows the user to change BSP or Category.

SYNOPSIS

```
#include "Change_RFQs_Form.H"

class Change_RFQs_Form: public DUI_Form { protected:
    Change_RFQs_Form();
    ~Change_RFQs_Form();
public:
    static Change_RFQs_Form *instance( RFQ_Summary * );
    void save_changes();
    void save_commit();
    void save_quit();
    Callback *save_commit_callback;
    Callback *save_quit_callback;
private:
    DUI_Selection *bsp;
    DUI_Selection *review_status;
    DUI_Field *rfq_count;
    DUI_Label *op_to_cl;
    DUI_Label *cl_to_op;
    DUI_Label *ch_to_op;
    static Change_RFQs_Form *instance_;
    RFQ_Summary *rfq_summary;
    void setup( RFQ_Summary * );
}
```

DESCRIPTION

This Class is instantiated by List\_RFQs\_Form::change\_selected\_rfqs() (see List\_RFQs\_Form(l)). It defines a form with two DUI\_Selection's (see DUI\_Selection(d)) from which the user can select the buyer and/or category that he wishes to apply to the rfqs selected in the RFQ\_Summary (see RFQ\_Summary(l)) passed in to the constructor. It actually performs the change by calling RFQ\_Summary::change\_selected\_rfqs().

MEMBER FUNCTIONS

Change\_RFQs\_Form::instance( RFQ\_Summary \*rfq\_summary )

Description: This function provides the only public access to the constructor. We want only one instance of this form active at one time. returns: a pointer to an instance of the form.

Change\_RFQs\_Form::Change\_RFQs\_Form()

Description: The private constructor called by instance(). returns: void

Change\_RFQs\_Form::~~Change\_RFQs\_Form()

Description: This destructor currently does nothing. returns: void

Change\_RFQs\_Form::save\_changes()

Description: Informs the user of what is about to be changed, and asks for confirmation. returns: void

Change\_RFQs\_Form::save\_commit()

Description: Actually commits the changes calling RFQ\_Summary::change\_selected\_rfqs() (See RFQ\_Summary(l)). returns: void

Change\_RFQs\_Form::save\_quit()

Description: This is called if the user presses quit on the save confirmation dialog. It does nothing. returns: void

Change\_RFQs\_Form::setup( RFQ\_Summary \*rfq\_sum )

Description: This routine sets up information according to the RFQ\_Summary passed as an argument. The options the user has on the category selection are limited depending on the contents of the RFQ\_Summary as follows: Closed -> Open (if there are any closed) Closed Held -> Open (if there are any closed held) Open -> Closed (if there are any open) This function is called by instance(). returns: void

## FILES

Change\_RFQs.C Change\_RFQs.H



### 2.3.4.2 List\_RFQs\_Form

#### NAME

List\_RFQs\_Form - This form displays the list of RFQs in an RFQ\_Summary.

#### SYNOPSIS

```
#include "List_RFQs_Form.H"
```

```
class List_RFQs_Form: public DUI_Form { protected:
    List_RFQs_Form();
    ~List_RFQs_Form();
public:
    static List_RFQs_Form *instance( RFQ_Summary * );
    void change_selected_rfqs();
    void view_statistics();
    void view_price_performance();
    void print_rfqs();
    void show_rfqs();
private:
    DUI_Label      *rfq_list_title;
    DUI_Multi_Selection *rfq_list;
    DUI_Field      *rfqs_shown;
    DUI_Field      *rfqs_in_list;
    static List_RFQs_Form *instance_;
    RFQ_Summary *rfq_summary;
    void setup( RFQ_Summary * );
    void setup_list(int just_add_more = 0);
    int propagate_selections();
}
```

#### DESCRIPTION

An RFQ\_Summary (see RFQ\_Summary(l)) is passed in to instance(), the contents of that summary are displayed in a DUI\_Selection (see DUI\_Selection(d)) then the following operations are allowed on RFQs selected from that list:

Change RFQs - see Change\_Form(l). View Statistics - see Statistics\_Form(l) Price Performance - see Price\_Performance\_Form(l) Print - see print\_rfqs() function.

#### MEMBER FUNCTIONS

List\_RFQs\_Form::instance( RFQ\_Summary \*rfq\_sum )

Description: This function provides the only public access to the constructor. We want only one instance of this form active at one time. returns: a pointer to an instance of the form.

List\_RFQs\_Form::List\_RFQs\_Form()

Description: The private constructor called by instance(). returns: void

List\_RFQs\_Form::~~List\_RFQs\_Form()

Description: This destructor currently does nothing. returns: void

List\_RFQs\_Form::show\_rfqs()

Description: This function is called when the user edits the "RFQs Shown" field on the List RFQs screen. If the number requested is greater than what is shown then the function just adds more from the list already queried. If it is less then it clears the list and lists just the ones asked for. (see setup\_list()) returns: void

List\_RFQs\_Form::change\_selected\_rfqs()

Description: Displays Change\_RFQs\_Form after synchronizing the RFQ\_Summary with the users selections (see propagate\_selections()). returns: void

List\_RFQs\_Form::view\_statistics()

Description: Displays Statistics\_Form after synchronizing the RFQ\_Summary with the users selections (see propagate\_selections()). returns: void

List\_RFQs\_Form::view\_price\_performance()

Description: Displays Statistics\_Form after synchronizing the RFQ\_Summary with the users selections (see propagate\_selections()). returns: void

List\_RFQs\_Form::print\_rfqs()

Description: Prints the current list of rfqs in the RFQ\_Summary. (see RFQ\_Summary::display\_data()). returns: void

List\_RFQs\_Form::setup( RFQ\_Summary \*rfq\_sum )

Description: This function is called by instance() it attaches the passed RFQ\_Summary and calls setup\_list() (which see). returns: void

List\_RFQs\_Form::setup\_list(int just\_add\_more)

Description: This function sets up the Mult\_Selection list and title based upon the number of rfqs requested and sort\_order implying what is to be displayed. If just\_add\_more is non-zero it just appends to the list until the number requested is satisfied else it clears the list and loads the number requested. returns: void

int List\_RFQs\_Form::propagate\_selections()

Description: This function propagates the selections the user has made to the internal RFQ\_Summary. It assumes that if the user has selected all the rfqs shown (which might not be the

actual number queried from the database) that he actually wanted to selected all the rfqs queried. returns: 1 if there were any selected by the user 0 otherwise.

## FILES

List\_RFQs.C List\_RFQs.H

### 2.3.4.3 Price\_History\_Form

#### NAME

Price\_History\_Form - Form to display the award-price history of a stock number.

#### SYNOPSIS

```
#include "Price_History_Form.H"
```

```
class Price_History_Form: public DUI_Form { protected:
    Price_History_Form();
    ~Price_History_Form();
public:
    static Price_History_Form *instance( RFQ_Summary * , char *,          double, double,
double, double);
    void print();
private:
    DUI_Field *stock_number;
    DUI_Table *price_table;
    DUI_Field *start_price;
    DUI_Field *end_price;
    DUI_Field *min_price;
    DUI_Field *max_price;
    static Price_History_Form *instance_;
    RFQ_Summary *rfq_summary;
    void setup( RFQ_Summary * ,char *, double, double, double, double);
}
```

#### DESCRIPTION

This form is instantiated by the Price\_Performance\_Form (which see). It displays a list of all prices at which this stock number was awarded as well as starting price, ending price, minimum price and maximum price.

#### MEMBER FUNCTIONS

number, Price\_History\_Form::instance( RFQ\_Summary \*rfq\_sum, char \*stock double startprice, double endprice, double minprice, double maxprice) Description: This function provides the only public access to the constructor. We want only one instance of this form active at one time. returns: a pointer to an instance of the form.

Price\_History\_Form::Price\_History\_Form()

Description: The private constructor called by instance(). returns: void

Price\_History\_Form::~~Price\_History\_Form()

Description: This destructor currently does nothing. returns: void

Price\_History\_Form::print()

Description: This function prints the current price history information using the environment variable "GATEC\_PRINT\_STRING". returns: void

number, Price\_History\_Form::setup( RFQ\_Summary \*rfq\_sum, char \*stock double startprice, double endprice, double minprice, double maxprice)

Description: This function is called by instance() and it finds the award entries in the RFQ\_Summary that was passed into the constructor and displays them in the history table. It also initializes the other fields. returns: void

## FILES

Price\_History.C Price\_History.H

#### 2.3.4.4 Price\_Performance\_Form

##### NAME

Price\_Performance\_Form - Form to display the awarded price changes by stock number.

##### SYNOPSIS

```
#include "Price_Performance_Form.H"
```

```
class Price_Performance_Form: public DUI_Form { protected:
    Price_Performance_Form();
    ~Price_Performance_Form();
public:
    static Price_Performance_Form *instance( RFQ_Summary * );
    void print();
    void view_price_history();
private:
    DUI_Selection *stock_numbers;
    static Price_Performance_Form *instance_;
    RFQ_Summary *rfq_summary;
    void setup( RFQ_Summary * );
    long *number_of_buys;
    long *start_date;
    long *end_date;
    double *max_price;
    double *min_price;
    double *start_price;
    double *end_price;
    List_of(STRING) stcknbrs;
}
```

##### DESCRIPTION

This Form displays the number of buys, starting date, ending date, maximum price, minimum price, starting price, and ending price for each awarded stock number represented in the RFQ\_Summary passed to it. It also allows a more specific history of a stock number to be viewed by instantiating a Price\_History\_Form.

##### MEMBER FUNCTIONS

Price\_Performance\_Form::instance( RFQ\_Summary \*rfq\_sum )

Description: This function provides the only public access to the constructor. We want only one instance of this form active at one time. returns: a pointer to an instance of the form.

Price\_Performance\_Form::Price\_Performance\_Form()

Description: The private constructor called by instance(). returns: void

Price\_Performance\_Form::~~Price\_Performance\_Form()  
Description: This destructor currently does nothing. returns:  
void

Price\_Performance\_Form::print()  
Description: This function prints the current performance table.  
returns: void

Price\_Performance\_Form::view\_price\_history()  
Description: This function instantiates the Price\_History\_Form  
for the selected stock number. returns: void

Price\_Performance\_Form::setup( RFQ\_Summary \*rfq\_sum )  
Description: This function does the actual calculations for each  
awarded stock number and populates the DUI\_Table (see  
DUI\_Table(d)) used to display it. returns: void

## FILES

Price\_Performance.C Price\_Performance.H

### 2.3.4.5 RFQ\_Summary

#### NAME

RFQ\_Summary - Class for querying and accessing a summary of GATEC RFQs.

#### SYNOPSIS

```
#include "RFQ_Summary.H"

class RFQ_Summary { private:
    Sort_Order *sort_order_;
    Summarized_RFQ **items_;
    Summarized_RFQ **selected_items_;
    Selection_Criteria *criteria_;
    int error_state_;
    String* last_error_;
    long count_;
    long selection_count_;
    long item_size_;
    long select_size_;
    List_ofPtrs(String) buyers;
    Connection * archive_;
    Connection * active_;
    static Sort_Order *current_order_;
    void add_selected(Summarized_RFQ * new_entry);
    void add_entry(Summarized_RFQ *new_entry, int arch);
    int find_entry(Summarized_RFQ *entry);
    void clear_entries();
    void clear_selections();
    int check_status(int type);
public:
    RFQ_Summary();
    ~RFQ_Summary();
    Summarized_RFQ *rfq(int i);
    Summarized_RFQ *selected_rfq(int i);
    int build_list() { return build_list(active_); };
    int build_list(Connection *con, int just_count = 0);
    void sort_list();
    void sort_selected();
    void select_rfq(int i);
    int change_selected_rfqs(const char *buyer = 0, const char *category = 0);
    int count() { return count_; };
    int query_count() { return build_list(active_, 1); };
    Lead Buyer(3) Last change: Tue Jan 4 16:19:33 1994      1

RFQ_Summary(3)      Gatec Manual      RFQ_Summary(3)
    int any_are_archived();
    int selection_count() { return selection_count_; };
```



```

void      set_selection_criteria(Selection_Criteria *new_criteria)
{ criteria_ = new_criteria; };
void set_sort_order(Sort_Order *new_order) { *sort_order_ = *new_order; };
Sort_Order *get_sort_order() { return sort_order_; };
int error() { return error_state_; };
char * error_msg() { return *last_error_; };
void select_all();
void deselect_all() { clear_selections(); };
void display_data(ostream& strm);
List_ofPtrs(String)* get_buyers() { return &buyers; };
enum {ALL_CLOSED = 1, SOME_CLOSED, NO_CLOSED,  ALL_OPEN, SOME_OPEN,
NO_OPEN,  ALL_HELD,  SOME_HELD,  NO_HELD,  ALL_AWARDED,
SOME_AWARDED, NO_AWARDED};
int check_held() { return check_status(ALL_HELD); };
int check_closed() { return check_status(ALL_CLOSED); };
int check_open() { return check_status(ALL_OPEN); };
int check_awarded() { return check_status(ALL_AWARDED);
};
public:
friend sort_summarized_rfqs(const void *t1, const void *t2);
}

```

## DESCRIPTION

This class allows the user to query the GATEC database for a list of rfqs based on the following criteria:

RFQ\_Number, Stock\_Number, Stock\_Class, SRAN,  
Review\_Status, RFQ\_Date, BSP

It keeps a list of Summarized\_RFQ's (see Summarized\_RFQ) populated from the database and allows the user to perform certain functions on the list.

## MEMBER FUNCTIONS

RFQ\_Summary::RFQ\_Summary()

Description: This is the only constructor defined, it initializes all member variables, establishes a default sort order, connects to the database and queries the Buyer table for a list of the current buyers. returns:  
void

RFQ\_Summary::~~RFQ\_Summary()

Description: Deletes storage and disconnects from database. returns: void

Summarized\_RFQ \*RFQ\_Summary::rfq(int i)

Description: Base List access function. returns:  
Sumarized\_RFQ \* indexed by i, if i is invalid it returns NULL.

Summarized\_RFQ \*RFQ\_Summary::selected\_rfq(int i)

Description: Selected List accessor function. The selected list is a list of pointers to objects in the base list that have been selected. returns:

Sumarized\_RFQ \* indexed by i, if i is invalid it returns NULL.

int RFQ\_Summary::build\_list(Connection \*con, int just\_count)

Description: This is the function that does the query and builds the list. It parses a its selection criteria using Range\_List, builds a ComplexQuery (see NARQ and NORA references) and calls add\_entry for each row returned. The argument con tells it which database connection to use (archive\_ or active\_), and argument just\_count tells it not to build a list but just to return the count. It is called by build\_list(void). returns: number of records retrieved.

void RFQ\_Summary::sort\_list()

Description: This function sorts the unselected list of items according to the sort\_order set by the user. It calls qsort() on the items\_list after setting the static variable current\_order\_ which is used by sort\_summarized\_rfqs(). returns: void

void RFQ\_Summary::sort\_selected()

Description: This function sorts the selected\_items\_list. returns: void

void RFQ\_Summary::select\_rfq(int i)

Description: Adds Summarized\_RFQ \* indexed by i from base list to selected list. returns: void

char \*category)

int RFQ\_Summary::change\_selected\_rfqs(const char \*buyer, const Description: Changes the database records associated with the list of selected RFQs to have the new buyer and or review status specified. It skips all the records that are from the archive database. returns: 0 always.

void RFQ\_Summary::add\_selected(Summarized\_RFQ \* new\_entry)

Description: Add an entry to selected RFQ's. returns: void

void RFQ\_Summary::add\_entry(Summarized\_RFQ \*new\_entry, int arch)

Description: add a brand new entry to list base list of Summarized\_RFQs, resizing if necessary. returns: void

int RFQ\_Summary::find\_entry(Summarized\_RFQ \*entry)

Description: Find an entry in base list equal to the argument

entry. returns: index of given argument or -1.

void RFQ\_Summary::clear\_entries()

Description: Remove all entries in the base list. returns: void

void RFQ\_Summary::clear\_selections()

Description: Remove all entries in selected rfq list. returns: void

int RFQ\_Summary::any\_are\_archived()

Description: Check to see if there are any archived entries in the selected list. returns: returns number of archive entries.

int sort\_summarized\_rfqs(const void \*t1, const void \*t2)

Description: Sort function used in call to qsort in sort\_selected and sort\_list. It uses the sort\_order in current\_order\_ to determine order. returns: -1 if t1 is greater than t2, 1 if t1 is less than t2, 0 otherwise.

void RFQ\_Summary::select\_all()

Description: Select all the in the base list rfqs. returns: void

void RFQ\_Summary::display\_data(ostream& strm)

Description: Print a summary of the criteria and the list that it generated onto the passed stream. returns: void

int RFQ\_Summary::check\_status(int type)

Description: Check to see if some, none or all of the selected rfqs are in the passed category. returns: one of the following: ALL\_CLOSED, SOME\_CLOSED, NO\_CLOSED, ALL\_OPEN, SOME\_OPEN, NO\_OPEN, ALL\_HELD, SOME\_HELD, NO\_HELD, ALL\_AWARDED, SOME\_AWARDED, NO\_AWARDED

## FILES

RFQ\_Summary.C RFQ\_Summary.H

### 2.3.4.6 Range\_List

#### NAME

Range\_List - a Class that handles parsing of a list of ranges in string form.

#### SYNOPSIS

```
#include "Range_List.H"
```

```
class Range { private:
    String * min_;
    String * max_;
public:
    Range(String& min, String& max) : min_(0), max_(0)
    { min_ = new String(min); max_ = new String(max); };
    ~Range() { delete max_; delete min_; };
    int single_value() { return (*min_ == *max_); };
    String& min() { return *min_; };
    String& max() { return *max_; };
};
List_of_Ptrsdeclare(Range)
class Range_List { private:
    List_ofPtrs(Range) ranges_;
    int hyphen_check;
public:
    Range_List(char *range_values, int hyphens = 1);
    ~Range_List() { ranges_.remove_all(); };
    int count() { return ranges_.size(); };
    Range * range(int i) { return ranges_[i]; };
}
```

#### DESCRIPTION

The Range\_List Class provides functionality for dealing with Ranges expressed as strings of the form:

[<value>[- <value>]], [<value>[- <value>]], ....

Single values are stored as ranges with min = max. This is inefficient but convenient and sufficient for small lists. The definition of Range is included with the definition of Range\_List.

#### MEMBER FUNCTIONS

Range\_List::Range\_List(char \*range\_values, int hyphen)

Description: The constructor parses the string passed to it (checking for "-" ranges if hyphen is no-zero) and generates the list of ranges. returns: void

## FILES

Range\_List.C Range\_List.H

### 2.3.4.7 Select\_RFQs\_Form

#### NAME

Select\_RFQs\_Form - Form querying the user for RFQ selection criteria.

#### SYNOPSIS

```
#include "Select_RFQs_Form.H"
```

```
class Select_RFQs_Form: public DUI_Form { public:
    Select_RFQs_Form();
    ~Select_RFQs_Form();
    void change_selection_order();
    void select_rfqs();
    void commit_select();
    void cancel_select();
    void quit();
    Callback *select_commit_callback;
    Callback *select_cancel_callback;
private:
    List_of(DUI_Toggle) select_toggles;
    DUI_Field *selection_order;
    DUI_Field *rfq;
    DUI_Field *stock_class;
    DUI_Text *stock_number;
    DUI_Field *sran;
    DUI_Field *start_date;
    DUI_Field * end_date;
    DUI_Group *criteria_group;
    DUI_Multi_Selection *buyer;
    DUI_Multi_Selection *review_status;
    List_of(DUI_Toggle) selected_toggles;
    Selection_Criteria *user_criteria;
    RFQ_Summary *rfq_summary;
    Sort_Order *user_order;
    enum Sort_Order::SortOption *order_type;
    void setup();
}
DESCRIPTION
```

This form provides a way for the user to enter criteria by which to build a list of rfqs to view statistics on or change. This is the first screen in the Lead Buyer application. The criteria are:

RFQ Number, Stock Class, Stock Number, Bill to SRAN, Date, Buyer, Review Status.

All criteria except date, and stock number can be entered as a list

in the form handled by Range\_List (which see).

This form instantiates the List\_RFQs\_Form (which see).

## MEMBER FUNCTIONS

Select\_RFQs\_Form::Select\_RFQs\_Form()

Description: Constructor for Select\_RFQs\_Form. returns: void

Select\_RFQs\_Form::~~Select\_RFQs\_Form()

Description: This destructor currently does nothing. returns: void

Select\_RFQs\_Form::change\_selection\_order()

Description: The criteria also has a sort order associated with it. This is represented by a set of DUI\_Toggles. By selecting and deselecting the toggles the user can change the sort order. This function is called whenever a user selects or deselects a toggle. returns: void

Select\_RFQs\_Form::select\_rfqs()

Description: This creates an RFQ\_Summary object with the given criteria and displays a List\_RFQs\_Form(which see) for that RFQ\_Summary. If the count returned by the RFQ\_Summary is > 5000 it asks the user if he wishes to continue. returns: void

Select\_RFQs\_Form::commit\_select()

Description: This function is called if the number of records that would be queried is over 5000 and the user decided to go ahead and generate the list. (See select\_rfqs()) returns: void

Select\_RFQs\_Form::cancel\_select()

Description: This function is called if the number of records that would be queried is over 5000 and the user decided to quit. It does nothing. returns: void

Select\_RFQs\_Form(3)

Gatec Manual

Select\_RFQs\_Form(3)

Select\_RFQs\_Form::quit()

Description: This function quits the lead buyer application altogether. returns: void

Select\_RFQs\_Form::setup()

Description: This function queries the buyer table for the list of buyers using RFQ\_Summary::get\_buyers() and updates the DUI\_Selection that lists them to the user. returns: void

## FILES

Select\_RFQs.C Select\_RFQs.H

### 2.3.4.8 Statistics\_Form

#### NAME

Statistics\_Form - Form to display RFQ statistics for an RFQ\_Summary.

#### SYNOPSIS

```
#include "Statistics_Form.H"
```

```
class Statistics_Form: public DUI_Form { protected:  
    Statistics_Form();  
    ~Statistics_Form();  
public:  
    static Statistics_Form *instance( RFQ_Summary * );  
    void print();  
private:  
    DUI_Table *table;  
    static Statistics_Form *instance_;  
    RFQ_Summary *rfq_summary;  
    void setup( RFQ_Summary * );  
}
```

#### DESCRIPTION

This form displays statistics about the selected RFQs in an RFQ\_Summary (which see). The statistics it displays are based on status and are:

Count in status, Dollar Amount in status, count percentage of total amount percentage of total.

#### MEMBER FUNCTIONS

Statistics\_Form::instance( RFQ\_Summary \*rfq\_sum )

Description: This function provides the only public access to the constructor. We want only one instance of this form active at one time. returns: a pointer to an instance of the form.

Statistics\_Form::Statistics\_Form()

Description: The private constructor called by instance(). returns: void

Statistics\_Form::~~Statistics\_Form()

Description: This destructor currently does nothing. returns: void

Statistics\_Form::print()

Description: Prints current statistics. returns: void



Statistics\_Form::setup( RFQ\_Summary \*rfq\_sum )  
Description: This funtion does that actual calculation of statistics.  
returns: void

## FILES

Statistics.C Statistics.H

### 2.3.4.9 Sort\_Order

#### NAME

Sort\_Order - Class for storing sorting orders to be applied to Summarized\_RFQ's.

#### SYNOPSIS

```
#include "Sort_Order.H"
```

```
class Sort_Order {
```

```
public:
```

```
    enum SortOption { RFQ_Number = 1, BSP, Stock_Number, Stock_Class, SRAN,  
Review_Status, RFQ_Date, RFQ_Quantity, RFQ_Price, Award_Quantity, Award_Price, Clear };
```

```
private:
```

```
    enum SortOption *sort_order;
```

```
    int next_;
```

```
    int find_opt(enum SortOption opt);
```

```
public:
```

```
    Sort_Order();
```

```
    ~Sort_Order();
```

```
    Sort_Order(Sort_Order& new_order);
```

```
    Sort_Order& operator = ( Sort_Order& new_order );
```

```
    Sort_Order& operator += ( enum SortOption opt );
```

```
    Sort_Order& operator -= ( enum SortOption opt );
```

```
    enum SortOption order(int which);
```

```
}
```

#### DESCRIPTION

This class is a utility class for storing and changing the order in which to sort the data that is stored in a Summarized\_RFQ (which see). It provides operators to easily modify the order.

#### MEMBER FUNCTIONS

Sort\_Order::Sort\_Order()

Description: initializes the sorting order to Clear(no order).  
returns: void

Sort\_Order::Sort\_Order(Sort\_Order& new\_order)

Description: Creates a Sort\_Order with the same elements as passed arg. returns: void

Sort\_Order::~~Sort\_Order()

Description: Deletes storage used by class. Namely the sort\_order array. returns: void

Sort\_Order& Sort\_Order::operator = ( Sort\_Order& new\_order )

Description: Assignment operator. returns: this Sort\_Order&.

Sort\_Order& Sort\_Order::operator += ( enum SortOption opt )

Description: if "opt" == Clear it clears the current option array else it adds "opt" to the end of the current option array. returns: this Sort\_Order&.

Sort\_Order& Sort\_Order::operator -= ( enum SortOption opt )

Description: if "opt" != Clear then it finds that option in the option array and removes it if it is there. returns: this Sort\_Order&.

enum Sort\_Order::SortOption Sort\_Order::order( int which )

Description: Finds order at index i. returns:  
Sort\_Option indexed by i.

int Sort\_Order::find\_opt( enum SortOption opt )

Description: Searches for opt in current array. returns: index of opt if there else Clear;

FILES

Sort\_Order.C Sort\_Order.H

### 2.3.4.10 String

#### NAME

String - A generic string class.

#### SYNOPSIS

```
#include "String.H"

class String {
protected:
    char *value_;
    int length_;
    int size_;
    static String *buf_;
    String();
private:
    void resize(int size);
    void set(const char *value, int len);
public:
    static String &buf();
    String(int size);
    String(String & );
    String(const char *str);
    String(const char *str, int length);
    virtual ~String();
    char *value() { return value_; };
    operator char *() { return value_; };
    int length() { return length_; };
    String &operator = (String & str);
    String &operator +=(String & str);
    String &operator = (const char *str);
    String &operator +=(const char *str);
    String &operator +=(char );
    boolean operator ==(String & str) { return (strcmp(value_, str.value_) == 0); };
    boolean operator !=(String & str) { return (strcmp(value_, str.value_) != 0); };
    boolean operator > (String & str) { return (strcmp(value_, str.value_) > 0); };
    boolean operator >=(String & str) { return (strcmp(value_, str.value_) >= 0); };
    boolean operator < (String & str) { return (strcmp(value_, str.value_) < 0); };
    boolean operator <=(String & str) { return (strcmp(value_, str.value_) <= 0); };
    boolean operator ==(const char *str) { return (strcmp(value_, str ? str : "" ) == 0); };
    boolean operator !=(const char *str) { return (strcmp(value_, str ? str : "" ) != 0); };
    char operator[](int index);
    boolean convert(int & );
    boolean convert(long & );
    boolean convert(float & );
    boolean convert(double & );
    void unjustify();
};
```

```

void center_justify(int length);
void right_justify(int length);
void left_justify(int length);
public:
virtual const char *class_name() const { return "String";
}
}

```

## DESCRIPTION

This class is a version of the DUI STRING class (which see) stripped of all the Communication\_Object functionality.

## MEMBER FUNCTIONS

String::String( int size )

Description: public constructors for String class String( int size ) - empty, null-terminated String of length size String( String & str ) - copy String String( char \*c ) - copy NULL-terminated array of char String( char \*c, int s ) - copy non NULL-terminated array of char returns: void

String::~~String()

Description: destructor for String returns: void

String::set( const char \*value, int size )

Description: sets value\_ to value and length\_ to length (growing String if needed) and NULL-terminates value\_ returns: void

String::resize( int size )

Description: resize value\_ if needed. returns: void

String& String::operator += (String & str)

Description: Concatenate str to end of String. returns: this String &.

String& String::operator += ( const char \* chars )

Description: Concatenates char \* chars to end of string. returns: this String&.

String& String::operator += ( char c )

Description: Concatenates char c to end of string.

Lead Buyer(3) Last change: Tue Jan 4 16:19:25 1994 2

String& String::operator = (String & str)

Description: assignment operator for String (from String). returns: this String &.

String& String::operator = (const char \* str)

Description: assignment operator for String (from char \*) returns: this String &.

String::operator[](int index)

Description: operator [n] returns the nth char in String returns: nth char.

String::convert(int &num)

Description: Converts string to int. returns: 1 if successful, 0 otherwise

String::convert(long &num)

Description: Converts string to long. returns: 1 if successful, 0 otherwise

String::convert(float &num)

Description: Converts string to float. returns: 1 if successful, 0 otherwise

String::convert(double &num)

Description: Converts string to double. returns: 1 if successful, 0 otherwise

String::unjustify()

Description: strips leading and trailing spaces. returns: void

String::left\_justify( int len )

Description: removes trailing spaces, pads with leading spaces. returns: void

String::center\_justify( int len )

Description: makes number trailing spaces = number leading spaces. returns: void

String(3)

Gatec Manual

String(3)

String::right\_justify( int len )

Description: removes leading spaces, pads with trailing spaces. returns: void

String::buf()

Description: This function allows access to static buf\_. returns: String & buf\_.

## FILES

String.C String.H

### 2.3.4.11 Summarized\_RFQ

#### NAME

Summarized\_RFQ - Class to hold data required on each RFQ in an RFQ\_Summary.

#### SYNOPSIS

```
#include "Summarized_RFQ.H"
```

```
class Summarized_RFQ{ private:
```

```
    String * RFQ_Number_;
    String * Line_Item_;
    String * BSP_;
    String * Stock_Number_;
    String * Stock_Class_;
    String * SRAN_;
    String * Review_Status_;
    String * RFQ_Date_;
    String * Award_Date_;
    String * UTN_Number_;
    String * Document_Id_;
    String * Redirect_Reason_;
    int  archived_;
    long  RFQ_JDate_;
    long  Award_JDate_;
    double Award_Price_;
    double RFQ_Price_;
    double Award_Quantity_;
    double RFQ_Quantity_;
```

```
public:
```

```
    Summarized_RFQ(const char * new_RFQ_Number_ = "", const char * new_Line_Item_ = "",
const char * new_BSP_ = "", const char * new_Stock_Number_ = "", const char *
new_Stock_Class_ = "", const char * new_SRAN_ = "", const char * new_Review_Status_ =
"", const char * new_RFQ_Date_ = "", const char * new_UTN_Number_ = "", const char *
new_Document_Id_ = "", const char * new_Redirect_Reason_ = "", const char *
new_Award_Date_ = "", double new_Award_Price_ = 0.0, double new_RFQ_Price_ =
0.0, double new_Award_Quantity_ = 0.0, double new_RFQ_Quantity_ = 0.0, long
new_RFQ_JDate_ = 0, long new_Award_JDate = 0);
```

```
    ~Summarized_RFQ();
    int archived() { return archived_; };
    void archived(int i) { archived_ = i; };
    void RFQ_Number(String& new_value);
    String& RFQ_Number();
    void Line_Item(String& new_value);
    String& Line_Item();
    void BSP(String& new_value);
    String& BSP();
    void Stock_Number(String& new_value);
```

```

String& Stock_Number();
void Stock_Class(String& new_value);
String& Stock_Class();
void SRAN(String& new_value);
String& SRAN();
void Review_Status(String& new_value);
String& Review_Status();
void RFQ_Date(String& new_value);
String& RFQ_Date();
void RFQ_Number(const char * new_value);
void Line_Item(const char * new_value);
void BSP(const char * new_value);
void Stock_Number(const char * new_value);
void Stock_Class(const char * new_value);
void SRAN(const char * new_value);
void Review_Status(const char * new_value);
void RFQ_Date(const char * new_value);
void Award_Date(const char * new_value);
void RFQ_JDate(long new_value);
void Award_JDate(long new_value);
char * UTN_Number() { return *UTN_Number_; };
char * Document_Id() { return *Document_Id_; };
char * Redirect_Reason() { return *Redirect_Reason_; };
String& Award_Date() { return *Award_Date_; };
long RFQ_JDate();
long Award_JDate();
void Award_Price(double new_value);
double Award_Price();
void RFQ_Price(double new_value);
double RFQ_Price();
void Award_Quantity(double new_value);
double Award_Quantity();
void RFQ_Quantity(double new_value);
double RFQ_Quantity();
virtual Summarized_RFQ& operator = (Summarized_RFQ& new_value);
virtual int operator == (Summarized_RFQ& new_value);
};
class Selection_Criteria: public Summarized_RFQ { public:
    Selection_Criteria(const char * RFQ_Range = "", const char *BSP_Range = "", const
char * Stock_Number_Range = "", const char * Stock_Class_Range = "", const char *
SRAN_Range = "", const char * Status_Range = "", const char * Date_Range = "") :
    Summarized_RFQ(RFQ_Range, "", BSP_Range, Stock_Number_Range,
Stock_Class_Range, SRAN_Range, Status_Range, Date_Range) {};
    char * RFQ_Range() { return RFQ_Number(); };
    char * BSP_Range() { return BSP(); };
    char * Stock_Number_Range() { return Stock_Number(); };
    char * Stock_Class_Range() { return Stock_Class(); };
    char * SRAN_Range() { return SRAN(); };
    char * Status_Range() { return Review_Status(); };
    char * Date_Range() { return RFQ_Date(); };

```



}

## DESCRIPTION

This is basically a structure with accessor functions that holds the data needed by the RFQ\_Summary class. Which is as follows:

RFQ Number, Line item, BSP, Stock Number, Stock Class, SRAN, Review Status, RFQ Date, UTN Number, Documentid, Redirect Reason, Award Date, Award price, RFQ price, Award quantity, RFQ Quantity, RFQ Date, Award date.

## MEMBER FUNCTIONS

```
Summarized_RFQ::Summarized_RFQ(const char *
new_RFQ_Number_, const char * new_Line_Item_,
const char * new_BSP_, const char *
new_Stock_Number_, const char *
new_Stock_Class_, const char * new_SRAN_,
const char * new_Review_Status_, const char *
new_RFQ_Date_, const char * new_UTN_Number_,
const char * new_Document_Id_, const char
* new_Redirect_Reason_, const char *
new_Award_Date_, double new_Award_Price_,
double new_RFQ_Price_, double
new_Award_Quantity_, double
new_RFQ_Quantity_, long new_RFQ_JDate_,
long new_Award_JDate_) Description: Creates a new
Summarized RFQ with passed data. returns: void
Summarized_RFQ::~~Summarized_RFQ()
Description: Destroys used storage. returns: void
```

```
void Summarized_RFQ::RFQ_Number(String& new_value)
Description: value seting function returns: void
```

```
String& Summarized_RFQ::RFQ_Number()
Description: Accessor function. returns: value
```

```
void Summarized_RFQ::Line_Item(String& new_value)
Description: value seting function returns: void
```

```
String& Summarized_RFQ::Line_Item()
Description: Accessor function. returns: value
```

```
void Summarized_RFQ::BSP(String& new_value)
Description: value seting function returns: void
```

```
String& Summarized_RFQ::BSP()
Description: Accessor function. returns: value
```

```
void Summarized_RFQ::Stock_Number(String& new_value)
Description: value seting function returns: value
```

String& Summarized\_RFQ::Stock\_Number()  
Description: Accessor function. returns: value

void Summarized\_RFQ::Stock\_Class(String& new\_value)  
Description: value seting function returns: void

String& Summarized\_RFQ::Stock\_Class()  
Description: Accessor function. returns: value

void Summarized\_RFQ::SRAN(String& new\_value)  
Description: value seting function returns: void

String& Summarized\_RFQ::SRAN()  
Description: value seting function returns: void

Summarized\_RFQ(3) Gatec Manual  
Summarized\_RFQ(3)

void Summarized\_RFQ::Review\_Status(String& new\_value)  
Description: value seting function returns: void

String& Summarized\_RFQ::Review\_Status()  
Description: Accessor function. returns: value

void Summarized\_RFQ::RFQ\_Date(String& new\_value)  
Description: value seting function returns: void

String& Summarized\_RFQ::RFQ\_Date()  
Description: Accessor function. returns: value

void Summarized\_RFQ::RFQ\_Number(const char \* new\_value)  
Description: value seting function returns: void

void Summarized\_RFQ::Line\_Item(const char \* new\_value)  
Description: value seting function returns: void

void Summarized\_RFQ::BSP(const char \* new\_value)  
Description: value seting function returns: void

void Summarized\_RFQ::Stock\_Number(const char \* new\_value)  
Description: value seting function returns: void

void Summarized\_RFQ::Stock\_Class(const char \* new\_value)  
Description: value seting function returns: void

void Summarized\_RFQ::SRAN(const char \* new\_value)  
Description: value seting function returns: void

void Summarized\_RFQ::Review\_Status(const char \* new\_value)  
Description: value seting function returns: void

void Summarized\_RFQ::RFQ\_Date(const char \* new\_value)  
Description: value seting function returns: void

void Summarized\_RFQ::Award\_Date(const char \* new\_value)  
Description: value seting function returns: void

void Summarized\_RFQ::RFQ\_JDate(long new\_value)  
Description: value seting function returns: void

void Summarized\_RFQ::Award\_JDate(long new\_value)  
Description: value seting function returns: void

long Summarized\_RFQ::RFQ\_JDate()  
Description: Accessor function. returns: value

long Summarized\_RFQ::Award\_JDate()  
Description: Accessor function. returns: value

void Summarized\_RFQ::Award\_Price(double new\_value)  
Description: Accessor function. returns: value

double Summarized\_RFQ::Award\_Price()  
Description: Accessor function. returns: value

void Summarized\_RFQ::RFQ\_Price(double new\_value)  
Description: Accessor function. returns: value

double Summarized\_RFQ::RFQ\_Price()  
Description: Accessor function. returns: value

void Summarized\_RFQ::Award\_Quantity(double new\_value)  
Description: Accessor function. returns: value

double Summarized\_RFQ::Award\_Quantity()  
Description: Accessor function. returns: value

void Summarized\_RFQ::RFQ\_Quantity(double new\_value)  
Description: Accessor function. returns: value

double Summarized\_RFQ::RFQ\_Quantity()  
Description: Accessor function. returns: value

&new\_value)  
Summarized\_RFQ & Summarized\_RFQ::operator =  
(Summarized\_RFQ Description: Assignment operator for  
Summarized\_RFQ. returns: this Summarized\_RFQ&.

Lead Buyer(3) Last change: Tue Jan 4 16:19:23 1994 6

Summarized\_RFQ(3) Gatec Manual  
Summarized\_RFQ(3)  
int Summarized\_RFQ::operator == (Summarized\_RFQ  
&new\_value)  
Description: Comparison operator for Sumarized\_RFQ. returns:  
1 if equal 0 if not.

#### FILES

Summarized\_RFQ.C Summarized\_RFQ.H

---

## 2.4 System Parameters Application

---

sys\_param.oui is an application that fulfills the user interface requirements for the system parameters functions of the GATEC project. It allows the user to view and modify the contents of tables which hold site specific operations data. The data that can be operated on are:

Award Piins  
Maximum Download Priority Price  
Gatec Buyers  
System Holidays  
Acknowledgement Type and Due Hours  
Quote response and delivery times by priority  
Email address for warning messages.

It uses DUI(1) for its user interface and interacts with a data base through the NARQ(see NARQ) and NORA(see NORA) libraries. It is written in C++. To get a user perspective on the sys\_param.oui application see the *System Parameters User's Guide* [REF000]

The following sections give a technical overview of the sys\_param.oui application.

---

### 2.4.1 Class Hierarchy

---

The sys\_param.oui application has the following class hierarchy, indentation denotes derivation:

(DUI\_Form) - defined in DUI(1)  
System\_Parameters\_Form

There is only one form in this application. It does all the querying and updating.

See the individual documentation on this class for more details.

---

### 2.4.2 Programming Hints

---

The documentation for the `System_Parameters_Form` class should be consulted, as well as the `DUI`, `NARQ` and `NORA` man pages because this will clarify a lot of the code found in the application.

The form makes extensive use of `DUI_Table(1)`s for displaying and editing the data base tables, so see the documentation for `DUI_Table` to understand how it is used.

The award piins are added through a `DUI_Dialog(1)` when the user requests adding new piins. The `DUI_Dialog` is more than just a simple pop-up so see its documentation for more detail.

None of the changes to the data base are committed until the user either requests a save or exits the application. This allows for the user to undo all the changes he has made up to the last save.

---

### 2.4.3 System Parameters Source Tree

---

The source for `sys_param.oui` is kept under the `DUI(1)` source tree in:

`$CVSROOT/oui/applications/sys_param`

It depends on the `NARQ` and `NORA` libraries being in:  
`$CVSROOT/narqdb/lib`

These must be made before the `sys_param.oui` application can be made. To make the `sys_param.oui` application, `cd` to its source directory and type:

`xmkmf; make depend all`

The resulting "`sys_param.oui`" file will be installed in:

`$CVSROOT/oui/bin`

---

### 2.4.4 System Parameters From Classes

---

As mentioned abovem only one form is used in the System Parameters application; specifically, `System_Parameters_Form`

### 2.4.4.1 System\_Parameters\_Form

NAME

System\_Parameters\_Form - Implements the System Parameters form used by sys\_param(which see).

SYNOPSIS

```
#include "System_Parameters_Form.H"
```

```
class System_Parameters_Form: public DUI_Form { public:
    System_Parameters_Form();
    ~System_Parameters_Form();
    void change_holiday_year();
    void add_piins_dialog();
    void add_piins();
    void add_piins_check_year();
    void save_changes();
    void undo_changes();
    void quit();
private:
    DUI_Field *download_max_priority;
    DUI_Field *download_max_price;
    DUI_Table *download_buyers;
    DUI_Table *piin_stats;
    DUI_Table *holidays;
    DUI_Field *holiday_year;
    DUI_Field *holiday_count;
    DUI_Table *response_times;
    DUI_Toggle *acknowledge_840;
    DUI_Toggle *acknowledge_850;
    DUI_Toggle *acknowledge_864;
    DUI_Field *acknowledge_due_hours;
    DUI_Field *warning_address;
    DUI_Group *add_piins_group;
    DUI_Toggle * gsa_piin;
    DUI_Field *start_piin;
    DUI_Field * end_piin;
    DUI_Field * piin_prefix;
    Callback *piin_dialog_callback;
    Callback *add_piins_callback;
    Callback *add_piins_check_callback;
    void setup();
    void setup_download();
    void setup_piins();
    void setup_holidays();
    void setup_response_times();
    void save_all();
    void save_download();
```

```

void save_holidays();
void save_response_times();
}

```

## DESCRIPTION

This class defines the main form used in the sys\_param application. It provides widgets necessary to perform the operations defined in sys\_param(which see). It deals with the following tables(see NARQ):

Piins, Buyer, UserManagerDefaults, PriorityGroup, Holidays

## MEMBER FUNCTIONS

System\_Parameters\_Form::System\_Parameters\_Form()

Description: Constructor for System\_Parameters\_Form. All of the widgets used in the application are instantiated here. returns: void

System\_Parameters\_Form::~~System\_Parameters\_Form()

Description: Destructor. Deletes add\_piins\_group, add\_piins\_callback, add\_piins\_check\_callback. returns: void

System\_Parameters\_Form::change\_holiday\_year()

Description: Changes holiday year when user enters new year. returns: void

System\_Parameters\_Form::add\_piins\_dialog()

Description: Prompts user for new piins constructing a DUI\_Dialog(which see) containing fields for the user to enter piin numbers and prefix. returns: void

System\_Parameters\_Form::add\_piins\_check\_year()

Description: Checks year on piin prefix and queries user if it is not this year. Also checks length of prefix to make sure it is 4 characters long, and notifies the user of an error otherwise. returns: void

System\_Parameters\_Form::add\_piins()

Description: Adds new piins to database. returns: void

System\_Parameters\_Form::save\_changes()

Description: Saves changes to database calling save\_all(). returns: void

System\_Parameters\_Form::undo\_changes()

Description: Rolls back database and call setup to reset the values on the screen. returns: void

System\_Parameters\_Form::quit()



Description: Quits the lead buyer application after committing the database. returns: void

System\_Parameters\_Form::setup()

Description: Sets up values on the form called by constructor. returns: void

System\_Parameters\_Form::setup\_download()

Description: Setup download criteria field values. returns: void

System\_Parameters\_Form::setup\_piins()

Description: Setup piin fields values. returns: void

System\_Parameters\_Form::setup\_holidays()

Description: Setup holiday field values. returns: void

System\_Parameters\_Form::setup\_response\_times()

Description: Setup response\_times field values. returns: void

System\_Parameters\_Form::save\_all()

Description: Save values. returns: void

System\_Parameters\_Form::save\_download()

Description: Save download criteria fields values. returns: void

System\_Parameters\_Form::save\_holidays()

Description: Save holiday field values. returns: void

System\_Parameters\_Form::save\_response\_times()

Description: Save response\_times field values. returns: void

## FILES

System\_Parameters.C System\_Parameters.H

---

## 2.5 Windui Application

---

windui implements DUI(1) client functionality for MS Windows 3.1. As for any DUI client, it implements the widgets defined by DUI in a manner consistent with the Windows operating system. To this end, its primary objectives were to find Windows 3.1 equivalents for displaying what is intended by the DUI widgets, and implement an appropriate communications path for attaching to the DUI server. See the BASIC IMPLEMENTATION STRATEGY section for details on how windui met these requirements and see DUI(1) for an explanation

The client uses the facilities of DUI for generating clients and the tools contained in Borland C++ and application frameworks 3.1 for creating the windows objects and compiling.

---

### 2.5.1 Basic Implementation Strategy

---

In order to establish a correspondence between windows interface elements and DUI interface widgets, windui defines a set of sister classes that correspond one to one with the DUI widgets. It makes modifications to the DUI widgets(using the facility provided by DUI) adding data and function members to create and update instances of the windui classes. They are derived from Borland's Object Window Library(OWL) classes (see Object Windows for C++ [REF000] ) and are described in more detail in the CLASS HIERARCHY section.

To meet the communications requirement, windui defines a new streambuf derivative, called SerialBuf(5), that implements serial port stream communications, and a class called Communications\_Script(5) which allows a rudimentary scripting language to be executed over a serial port. Currently this is the only communications it supports. It establishes a connection to the remote server by logging in to the remote machine over a serial line and executing the server from the login shell. The "logging in" is done by running a communications script(which can be written to do what ever is necessary to get to the remote machine, be it dialing a modem or connecting to a terminal server and issuing a telnet command). Windui retrieves the name of the application it is going to execute from the first argument on its command line, and retrieves the search path from a variable called "APP\_PATH" in its shell(DOS) environment. See the COMMUNICATIONS section for details about the scripting language.

The last implementation consideration arises from the desire to make the windui client as generic as possible and to give the user as much power as possible in determining how the forms will display. To this end, the windui program reads a file called "wres.res" in its current directory. It expects this file to contain a set of format specifications. These are called DUI resources and are modeled after XWindow resources. They are discussed in more detail in the DUI RESOURCES section. These allow the user to determine the size, placement and other visual aspects of the forms displayed by DUI applications.

---

### 2.5.2 Class Hierarchy

---

Windui has the following class hierarchy, indentation denotes derivation:

```
Device_Independent_Bitmap
  Pushbutton_Bitmap
Local_Atom
Table_String
SerialBuf
(TDialog)
  Prompt_Dialog
  Communications_Script
(TWindow) - (WS_OVERLAPPEDWINDOW)
  TMainWindow
  WTWindow
(TButton) - BUTTON(BS_PUSHBUTTON)
  WTButton
(TCheckBox) - BUTTON(BS_CHECKBOX)
  WTCheckBox
(TComboBox) - COMBOBOX
  WTComboBox
(TEdit) - EDIT
  WTEdit
  WText
(TGroupBox) - (WS_GROUP)
  WTGroupBox
(TListBox) - LISTBOX
  WTLListBox
  WTable
(TRadioButton) - BUTTON(BS_RADIOBUTTON)
  WTRadioButton
(TStatic) - STATIC
  WTStatic
```

The classes in ()'s are defined in Borland's Object Windows Library (OWL). They are C++ implementations of the Windows' predefined window classes. The name of the actual windows' class and style is given next to the its Borland equivalent. All of the classes prefixed with "WT" are windui sister classes. They are associated with their DUI relatives as follows:

```
WTWindow    w_View
WTButton     w_Command
WTCheckBox   w_Toggle
WTComboBox   w_Selection
WTEdit       w_Field
WText        w_Text
WTGroupBox   w_Group
WTLListBox   w_Selection
WTable       w_Table
```

WTRadioButton      w\_Toggle  
WTStatic          w\_Label

You will notice that the "DUI\_" prefix has been replaced with the "w\_" prefix for the names of the DUI classes. This is a result of the code generation for DUI clients (see DUI(1)). You will also notice that some DUI widgets have more than one sister class. This is because some of the widgets can have more than one representation. The representation used is determined by the resources for the form (see DUI RESOURCES section).

The support classes:

Device\_Independent\_Bitmap  
  Pushbutton\_Bitmap  
Local\_Atom  
Table\_String  
SerialBuf  
(TDialog)  
  Prompt\_Dialog  
  Communications\_Script

are not related directly to DUI, but perform functions specific to the client. The two bitmap classes allow for the user to specify bitmaps (using resources) to represent commands. The Local\_Atom class is a C++ implementation of Windows' ATOM's and is used to store resource statements. The Prompt\_Dialog, Communications\_Script, and SerialBuf classes are used by the serial communications package.

Each of the windui classes has its own individual documentation which should be consulted for more detail. There is also documentation on the client extensions made to the original DUI classes. It is under the modified class name (e.g. w\_Field(5)) will give the client extensions made to DUI\_Field).

---

### 2.5.3          DUI Resources

---

The word "resources" in this context does not refer to Windows' resources which are an integral part of the Windows' operating system. DUI resources are textual lines of the form:

<path>.<resource name>: <resource value>

They are read by the client from a file called "wres.res" at startup and are referenced throughout the session by the widgets instantiated during the session for formatting and representation

information.

The <path> component of the resource statement can have the following forms:

Absolute path:

Each element in the dot "." separated list of elements is either a widget name(i.e. the name of that particular instance of a widget) or a class name if the widget has no name (e.g. "Group" if the widget is a w\_Group). This path takes first precedence in the case of path conflicts. Thus:

<widget name | widget class name>[. <widget name | widget class name>]...

example 1: Main Form.First Group.First Field.length: 6

example 2: Main Form.Group.First Field.length: 6

View relative path:

This path includes only a view name, and a widget name. It refers to any widget with this name on the named view. It takes second precedence in conflicts. Thus:

<view name>\*<widget name>

example 1: Main Form\*First Field.length: 6

Relative path:

This path takes just a widget name. It refers to any widget with this name. It takes third precedence. Thus:

\*<widget name>

example 1: \*First Field.length: 6

View-class name path:

This takes a view name and a class name. It refers to any widget of this class type on the named view. It takes fourth precedence. Thus:

<view name>\*<class name>

example 1: Main Form\*Field.length: 6

Class name path:

This takes just a class name. It refers to any widget with this class. It takes fifth precedence. Thus:

<class name>

example 1: Field:length: 6

#### Defaults:

All widgets have a default for each of the resources they interpret.

A detailed list of the resource names and their meaning are given below by widget.

#### View:

name - Any textual value  
(the name of the widget).

Accepts no other resource modifications.

#### All Widgets (except views):

name - Any textual value (the name of the widget).  
fontname - name of a type face.  
fontheight - font height metric (numeric)  
fontfixed - is font fixed width ("yes", "no").  
fontunderline - is font underlined ("yes", "no").  
fontitalic - is font italic ("yes", "no").  
fontweight - how bold is font (numeric).

#### Field:

defaultvalue- initial value to give field if it has no value.  
length - length of field in "M" characters of chosen font.  
(numeric)

#### Text:

defaultvalue- initial value to give field if it has no value.  
length - length of field in "M" characters of chosen font.  
(numeric)  
itemssshown - Number of lines to show (numeric).  
width - width in pixels (numeric).  
height- height in pixels (numeric).  
waitedfor - stop waiting when this is received ("yes",  
"no").

#### Selection:

defaultvalue - initial value to give field if it has no value.  
length - length of field in "M" characters of chosen font.  
(numeric)

itemsshown - Number of lines to show (numeric).  
width - width in pixels (numeric).  
height- height in pixels (numeric).  
wait - wait for another widget to be sent? ("yes", "No").

Group:

layout - direction to lay out widgets ingroup ("horizontal", "vertical"). dimensions - number of widgets horizontally and vertically (dimension e.g. 2x3).  
horizontalspacing - number of pixels between widgets horizontally. (numeric)  
verticalspacing- number of pixels between widgets vertically. (numeric)  
explicitdimensions - number of widgets to lay out in each row. (space separated list of numbers e.g. 3 4 1 5).

Toggle:

defaultvalue - initial value to give field if it has no value.  
representation - type of toggle ("radio", "check").  
length - length of field in "M" characters of chosen font. (numeric)

Command:

verticalspacing - for groups of buttons, number of pixels between elements vertically (numeric).  
length - length of field in "M" characters of chosen font. (numeric)  
bitmap - file name of Windows DIB to use as button representation (string, full path name of bitmap file.).  
width - width in pixels (numeric).  
height - height in pixels (numeric).

The code for accessing these resources is in w\_widget.cc and w\_widget.hh (the extensions to the DUI Widget class). It is kept here because it needs to be inherited by all other widgets.

---

## 2.5.4 Communications

---

As stated above windui communicates with the remote server through a serial port. It retrieves the serial port configuration information from a file called "com.cfg" which is modified using an auxiliary windui program called "setup.exe". This program must be run to set up the serial port configuration information

and script file that will be used to establish the connection. It has options for data bits, stop bits, baud rate, parity and connection type. The connection type is the switch that specifies what script file to execute. Connection type has three options, network, modem, direct. These are associated with script files "network.scr", "modem.scr", and "direct.scr" in the current directory, and can be written to do anything the script writer needs to do to establish a connection.

The scripting language consists of the following commands, which must start at the beginning of a line:

transmit <value up to new line> waitfor <timeout in 100th's of a second> <value up to new line> pause <time to pause in 100th's of a second>

In addition the "transmit" can have the following key words as its value:

ATDTTELEPHONENUMBER RETURN USERID PASSWORD

The ATDTTELEPHONENUMBER causes a prompt to the user asking for a telephone number which is tacked on to the end an "atdt" string and sent to the port. RETURN causes a blank line to be sent. USERID and PASSWORD both prompt the user for an entry and send the entered string to the serial port as typed.

The script can have any number of blank lines and lines that begin with "#" (comments). These are ignored.

In addition to the script used to connect, there is a script used to disconnect. This script is called "discon.scr". It is run no matter which connection type is selected.

Some modifications were made to the DUI Session class in order to support Windows serial communication. DUI provides no facility for modifying this class so a separate copy of the file with modification is kept in the windui source tree. See Session(5) for more details about the modifications.

---

### 2.5.5 WINDUI Source Directory

---

The source for the windui client is kept in the DUI tree. It is not compiled there but has source dependencies on DUI libraries' source. It is kept in:

\$CVSROOT/dui/src/clients/win3.1



The DUI libraries must be made first before the windui client. Once they are made, the windui client can be made by going to the source directory and typing:

```
xmkmf; make windui
```

This collects the source it needs from the DUI libraries, changes file names to those suitable for DOS, and copies all the changed source files to a directory:

```
$CVSROOT/dui/src/clients/win3.1/changed
```

The files in this directory are DOS files and can be copied to a Windows machine and compiled using Borland C++ 3.1.

---

## 2.5.6 WINDUI Classes

---

The following classes/objects are made use of in the windui application:

- Communications\_Script
- Device\_Independent\_Bitmap
- Local\_Atom
- Prompt\_Dialog
- Pushbutton\_Bitmap
- TMainWindow
- Table\_String
- WTButton
- WTCheckBox
- WTComboBox
- WTEdit
- WTGroupBox
- WTListBox
- WTRadioButton
- WTStatic
- WTText
- WTWindow
- WTable
- w\_Command
- w\_Component
- w\_End\_Command
- w\_Field
- w\_Group
- w\_Label
- w\_Selection
- w\_Table
- w\_Text
- w\_Toggle

w\_View  
w\_Widget  
Session  
SerialBuf

They are described in the following sections.

### 2.5.6.1 Communications\_Script

NAME

- class for running a communications script file on a stream.

SYNOPSIS

```
#include "Communications_Script.H"
```

```
class Communications_Script: public TDialog { private:
```

```
    iostream *iostream_;
```

```
    STRING script_;
```

```
    ifstream *scriptfile_;
```

```
    STRING error_;
```

```
    STRING line_;
```

```
    STRING value_;
```

```
    LONG time_out_;
```

```
    int timer_;
```

```
    char *compstring_;
```

```
public:
```

```
    Communications_Script(PWindowsObject AParent, iostream *thisiostream, char  
*newscript = 0, int real = 0 );
```

```
    ~Communications_Script();
```

```
    int run_script();
```

```
    void script(char *nscript);
```

```
    char *script() { return script_; };
```

```
    char *error() { return error_; };
```

```
    virtual void SetupWindow();
```

```
    virtual void WMCommand(RTMessage Msg) = [WM_FIRST + WM_COMMAND];
```

```
    virtual void WMTimer(RTMessage Msg) = [WM_FIRST + WM_TIMER];
```

```
    virtual void WMClose(RTMessage Msg) = [WM_FIRST + WM_CLOSE];
```

```
    virtual void WMSetCursor(RTMessage Msg) = [WM_FIRST + WM_SETCURSOR];
```

```
protected:
```

```
    void ProcessNextCommand();
```

```
    void WaitForCommand();
```

```
    void TransmitCommand();
```

```
    void PauseCommand();
```

```
void ShiftCompString();
```

```
private:
```

```
Communications_Script *real_dialog_;
```

```
int real_;
```

```
void setreal() { real_=1; };
```

}

## DESCRIPTION

This class is used to run a script of io operations on a stream inside a windows application. It pops up an invisible dialog and runs commands of the following form from a script file:

Waitfor <timeout in seconds> <string to waitfor> Transmit  
<string to transmit> Pause <time to pause>

It uses Window's timers to process timeouts and pauses.

## MEMBER FUNCTIONS

AParent,  
Communications\_Script::Communications\_Script(PTWindowsObject  
iostream \*thisiostream, char  
\*newscript, int real ) Description: Constructor for  
Communications\_Script. returns: void

void Communications\_Script::script(char \*nscript)  
Description: Sets the script file to be processed. returns: void

int Communications\_Script::run\_script()  
Description: This function actually runs the script. To facilitate  
error message passing this function creates a duplicate  
Communications\_Script object with "real" set to true and  
executes that dialog. This is because dialogs are deleted when they  
terminate. returns: void

Communications\_Script::~Communications\_Script()  
Description: Destructor. deletes memory and file descriptors.  
returns: void

void Communications\_Script::WMCommand(RTMessage Msg)  
Description: Lines are read from the script file and processed  
using messages. this script file can have blank lines and comments  
beginning with The allowed commands are: Waitfor <timeout  
in seconds> <string to waitfor> Transmit <string to transmit>  
Pause <time to pause> returns: void

void Communications\_Script::WMTimer(RTMessage Msg)  
Description: This is the function that is called when the dialog  
receives a timer message. A timer is created for "pause" and  
"waitfor" commands. A timer message is sent at .01 second  
intervals. If it is a pause timer message then time is simply counted  
down, if it is a waitfor timer, then characters are read from the  
stream and concatenated onto a test string which is shifted one  
character every time a character is read and this string is compared  
to the expected value and if it matches before the timeout is

reach, the next command is processed else a timeout is signified.  
returns:  
void

void Communications\_Script::WMClose(RTMessage Msg)  
Description: Called when the window is closed. It kills any left over timers and calls TDialog::WMClose(). returns: void

void Communications\_Script::WMSetCursor(RTMessage Msg)  
Description: this function is called when the window receives a WM\_SETCURSOR message. It calls the applications main window cycle\_cursor function. This is for use with the GATEC application specifically and should be removed for generic use.  
returns: void

void Communications\_Script::SetupWindow()  
Description: This function is called when the dialog is executed. It moves the dialog window off the visible screen and makes it 0 length and 0 width so it will be invisible. returns: void

void Communications\_Script::ProcessNextCommand()  
Description: This function sends a WM\_COMMAND message with a PROCESS\_NEXT\_COMMAND argument, so the WMCommand() function will read the next command from the script file. returns: void

void Communications\_Script::WaitForCommand()  
Description: This function sends a WM\_COMMAND message with a WAIT\_FOR\_COMMAND argument, so the WMCommand() function perform a waitfor command. returns: void

Description:  
This function sends a WM\_COMMAND message with a TRANSMIT\_COMMAND argument, so the WMCommand() function perform a transmit command. returns: void

Description:  
This function sends a WM\_COMMAND message with a PAUSE\_COMMAND argument, so the WMCommand() function will perform a pause command. returns: void

void Communications\_Script::ShiftCompString()  
Description: This function shifts the test string to the left by one character. returns: void

## FILES

comscr.cpp comscr.hpp

## 2.5.6.2 Device\_Independent\_Bitmap

### NAME

Device\_Independent\_Bitmap - Class encapsulating Windows DIB's.

### SYNOPSIS

```
#include "Device_Independent_Bitmap.H"
```

```
class Device_Independent_Bitmap { private:
    char *pixeldata_;
    HBITMAP bitmap_handle_;
    BITMAPFILEHEADER bitmap_file_header_;
    BITMAPINFOHEADER bitmap_info_header_;
    char *info_header_and_RGB_info_;
    HDC resize_memory_DC_1;
    int status_;
protected:
public:
    Device_Independent_Bitmap();
    Device_Independent_Bitmap(char *file_name);
    ~Device_Independent_Bitmap();
    HBITMAP bitmap_handle() { return bitmap_handle_; };
    void load_from_file(char *file_name);
    int delete_bitmap();
    int create_for_context(HDC destination_dc);
    int status() { return status_; };
    BITMAPINFOHEADER * bitmap_info_header() { return &bitmap_info_header_; };
    long bitmap_width() { return bitmap_info_header_.biWidth;
};
    long    bitmap_height()    {    return bitmap_info_header_.biHeight; };
}
```

### DESCRIPTION

This class provides functionality for dealing with Window's DIB's. It provides methods for reading DIB descriptions from a file and creating a DIB image for a supplied device context.

### MEMBER FUNCTIONS

Device\_Independent\_Bitmap::Device\_Independent\_Bitmap()  
Description: Empty Constructor. Creates an empty DIB. returns:  
void

\*file\_name)

Device\_Independent\_Bitmap::Device\_Independent\_Bitmap(char

Windui(5) Last change: Wed Jan 5 18:03:40 1994

1

Device\_Independent\_Bitmap(5) Gatec

Manual Device\_Independent\_Bitmap(5)

Description: Constructor accepting file name as an argument. It reads in the DIB description from the file expecting it to be in DIB format. returns: void

Device\_Independent\_Bitmap::~Device\_Independent\_Bitmap()

Description: Destructor. Cleans up memory usage. returns: void

void Device\_Independent\_Bitmap::load\_from\_file(char \*file\_name)

Description: Loads a new bitmap from the file named by arg "file\_name". returns: void

destination\_dc)

int Device\_Independent\_Bitmap::create\_for\_context(HDC

Description: Creates a DIB for the passed device context. returns: int, -1 if failure, 0 if success.

int Device\_Independent\_Bitmap::delete\_bitmap()

Description: Deletes bitmap handle if one was created. returns: int 0 if success, -1 if failure.

## FILES

bitmap.cpp bitmap.hpp

### 2.5.6.3 Local\_Atom

#### NAME

Local\_Atom - Class encapsulating Windows 3.1 Atoms.

#### SYNOPSIS

```
#include "Local_Atom.H"

class Local_Atom: public Object { private:
    ATOM atom_handle_;
    char *atom_value_;
public:
    Local_Atom();
    Local_Atom(char *new_atom);
    Local_Atom(ATOM new_atom);
    ~Local_Atom();
    /* pure virtual functions needing definition from Object. */ Local_Atom(Local_Atom&
new_atom);
    virtual hashValueType hashValue() const ;
    virtual classType isA() const;
    virtual int isEqual( const Object& testObject) const ;
    virtual char *nameOf() const ;
    virtual void printOn(ostream& outputStream) const ;
    virtual int isAssociation() const { return 0; }
    /* additional useful behaviours. */

    char *atom_value();
    ATOM atom_handle();
    int valid() const { return (atom_handle_ == 0 ? 0 : 1);
};
}
```

#### DESCRIPTION

This class is used for storing values in Windows 3.1 Local Atoms which are entries in a systemic hash table.

#### MEMBER FUNCTIONS

Local\_Atom::Local\_Atom(char \*new\_atom)

Description: Constructor for atom entry. install new character string into the local atom table. returns: void

Local\_Atom::Local\_Atom(Local\_Atom& new\_atom)

Description: copy constructor for atom entry. install new character string into atom table. returns: void

Local\_Atom(5)

Gatec Manual

Local\_Atom(5)

Local\_Atom::Local\_Atom(ATOM new\_atom)  
Description: Constructor for atom entry. attach atom if there is an associated string in the atom table. returns: void

Local\_Atom::Local\_Atom()  
Description: Constructor for atom entry. Create an invalid Local\_Atom Object. returns: void

Local\_Atom::~~Local\_Atom()  
Description: Destructor for atom entry. Remove an atom entry from the local atom table. returns: void

char \*Local\_Atom::atom\_value()  
Description: Retrieve atom value. returns: void

ATOM Local\_Atom::atom\_handle()  
Description: Retrieve atom handle returns: ATOM, the handle.

hashValueType Local\_Atom::hashValue()  
Description: A Function that need to be defined for a derivative of Object. returns: hashValueType, the value.

classType Local\_Atom::isA()  
Description: Identifier function. Must be defined by a derivative of object. returns: classType, localatom- Class always.

int Local\_Atom::isEqual( const Object& testObject)  
Description: Must be defined by a derivative of Object. returns: int 1 if equal, 0 otherwise.

char \*Local\_Atom::nameOf()  
Description: Must be defined by a derivative of Object. returns: char \*, "Local\_Atom" always.

void Local\_Atom::printOn(ostream& outputStream)  
Description: Prints the value and handle labeled appropriately onto "outputStream". returns: void

## FILES

atomcl.cpp atomcl.hpp



## 2.5.6.4 Prompt\_Dialog

### NAME

Prompt\_Dialog - generic prompt dialog.

### SYNOPSIS

```
#include "Prompt_Dialog.H"

class Prompt_Dialog: public TDialog {

public:
    Prompt_Dialog(PWindowsObject      AParent,      char *prompt_string,  char
**entered_value, int max, int hidden = 0);
    ~Prompt_Dialog();
    virtual void SetupWindow();
    virtual void WMCommand(RTMessage Msg) = [WM_FIRST + WM_COMMAND];
private:
    char **entered_value_;
    char *prompt_string_;
    int max_;
    int hidden_;
}
```

### DESCRIPTION

This is a generic dialog for popping up a single field dialog using a passed in prompt. It is used by Communications\_Script when certain transmission keywords are used.

### MEMBER FUNCTIONS

\*prompt\_string,  
Prompt\_Dialog::Prompt\_Dialog(PWindowsObject AParent,  
char char \*\*entered\_value, int max, int hidden)  
Description: Constructor. It accepts a prompt string which will appear on the dialog, a char \*\* which will contain the string that the user enters, and a length for the entered\_string value. returns: void

Prompt\_Dialog::~~Prompt\_Dialog()  
Description: Destructor. Does nothing. returns: void

void Prompt\_Dialog::SetupWindow()  
Description: This function sets the prompt string for this dialog. returns: void

Prompt\_Dialog(5) Gatec Manual Prompt\_Dialog(5)  
void Prompt\_Dialog::WMCommand(RTMessage Msg)  
Description: This function retrieves the value entered by the user

and then closes down the window. returns:  
void

## FILES

prompt.cpp prompt.hpp

### 2.5.6.5 Pushbutton\_Bitmap

#### NAME

Pushbutton\_Bitmap - class for displaying a bitmap as a push button.

#### SYNOPSIS

```
#include "Pushbutton_Bitmap.H"
```

```
class Pushbutton_Bitmap : public Device_Independent_Bitmap { private:
    HDC temporary_dc_;
    POINT *polygon_;
public:
    Pushbutton_Bitmap(char * file_name = 0);
    ~Pushbutton_Bitmap();
    int pushbutton_width() { return bitmap_width()+10; };
    int pushbutton_height() { return bitmap_height()+10; };
    void selected(HDC destination_hdc, int dX, int dY, int dH, int dW);
    void pushed(HDC destination_hdc, int dX, int dY, int dH, int dW);
    void unselected(HDC destination_hdc, int dX, int dY, int dH, int dW);
    void disabled(HDC destination_hdc, int dX, int dY, int dH, int dW);
    void draw_selection_line(HDC destination_hdc, int dX, int dY, int dH, int dW, int
is_selected);
    void draw_border(HDC destination_hdc, int dX, int dY, int dH, int dW);
    void upperleft_polygon(HDC destination_hdc, int dX, int dY, int dH, int dW, int
is_shadow);
    void lowerright_polygon(HDC destination_hdc, int dX, int dY, int dH, int dW, int
is_shadow);
}
```

#### DESCRIPTION

This class is used to create a push button out of bitmap. It draws edges on the outside of the bitmap and allows for these to be changed reflecting the following states:

selected, pushed, unselected, disabled

It is derived from Device\_Independent\_Bitmap(which see).

#### MEMBER FUNCTIONS

Pushbutton\_Bitmap::Pushbutton\_Bitmap(char \* file\_name )

Description: Constructor accepting filename as argument which is passed to constructor for base class Device\_Independent\_Bitmap. returns: void

Pushbutton\_Bitmap::~~Pushbutton\_Bitmap()

Description: Destructor. It does nothing. returns:

void  
dY, void Pushbutton\_Bitmap::selected(HDC destination\_hdc, int  
dX, int int dH, int dW) Description: This function paints  
the bitmap unto "destination\_hdc", and draws it such that it has a  
bold outline(2 pixel black border) which signifies selection.  
returns: void

dY, void Pushbutton\_Bitmap::pushed(HDC destination\_hdc, int  
dX, int int dH, int dW) Description: This function  
paints the bitmap onto "destination\_hdc" such that it is selected()  
and has button edges whose shading is reversed which signifies  
pushed. returns: void

int dY, void Pushbutton\_Bitmap::unselected(HDC  
destination\_hdc, int dX, int dH, int dW) Description:  
This function draws the bitmap with a plain (1 pixel black) border.  
Which signifies unselected (the default condition). returns: void

dY, void Pushbutton\_Bitmap::disabled(HDC destination\_hdc, int  
dX, int int dH, int dW) Description: There is no  
"disabled" view at present so this function is identical to  
unselected(). returns: void

int dX, void Pushbutton\_Bitmap::draw\_selection\_line(HDC  
destination\_hdc, int dY, int dH, int dW,  
int is\_selected) Description: This function draws the 2 pixel  
selection border in black if it is selected or in the background  
window color if it is unselected. returns: void

void Pushbutton\_Bitmap::draw\_border(HDC destination\_hdc, int  
dX, int dY, int dH, int dW)  
Description: This function draws the 1 pixel border around the  
bitmap. It is always drawn. returns: void

int dX, void Pushbutton\_Bitmap::upperleft\_polygon(HDC  
destination\_hdc, int dY, int dH, int dW,  
int is\_shadow) Description: This function draws the upper left  
polygon around the top and left sides of the bitmap. If the button is  
pushed this is shaded otherwise it is white. returns:  
void

int dX, void Pushbutton\_Bitmap::lowerright\_polygon(HDC  
destination\_hdc, int dY, int dH, int dW,  
int is\_shadow) Description:  
This function draws the lower right polygon around the bottom  
and right sides of the bitmap. If the button is pushed this is white  
otherwise it is shaded. returns:  
void

## FILES

buttbmp.cpp buttbmp.hpp

### 2.5.6.6 TMainWindow

#### NAME

TMainWindow - main window for windui.

#### SYNOPSIS

```
#include "TMainWindow.H"
```

```
class TMainWindow : public TWindow { public:  
    TMainWindow(PAllWindowsObject AParent, LPSTR ATitle, char *appname);  
    ~TMainWindow();  
    virtual void SetupWindow();  
    virtual void WMSetCursor(RTMessage Msg) = [WM_FIRST + WM_SETCURSOR];  
    char *appname;  
    int cycle_cursor();  
}
```

#### DESCRIPTION

This class defines the main window for Windui(which see). This window stays minimized for a windui session.

#### MEMBER FUNCTIONS

char \*nappname)

TMainWindow::TMainWindow(PAllWindowsObject AParent, LPSTR ATitle, Description: Define TMainWindow, a TWindow constructor. returns: void

TMainWindow::~~TMainWindow()

Description: Destructor. returns: void

void TMainWindow::SetupWindow()

Description: Establishes a Client\_Session. returns: void

void TMainWindow::WMSetCursor(RTMessage Msg)

Description: Responds to WM\_SETCURSOR message by calling cycle\_cursor(). returns: void

int TMainWindow::cycle\_cursor()

Description: Cycles through one of the seven states each time it is called displaying the cursor associated with that state. This is called each time the client goes into a waiting condition. It is used to animate the hourglass cursor. returns: int, 1 if changed, 0

Windui(5) Last change: Wed Jan 5 18:02:39 1994 1

TMainWindow(5) Gatec Manual TMainWindow(5)  
otherwise.

## FILES

tmainwin.cpp tmainwin.hpp

### 2.5.6.7 Table\_String

#### NAME

Table\_String - string class used by WTable.

#### SYNOPSIS

```
#include "Table_String.H"

class Table_String { private:
    int starting_position_;
    int insert_position_;
    char *string_;
    int length_;
public:
    Table_String(char *value = "");
    Table_String(Table_String& value);
    ~Table_String();
    void start_position(int pos_vector)
    { starting_position_ += pos_vector;
      if (starting_position_ >= length_)
        starting_position_ = length_-1;
      if (starting_position_ < 0)
        starting_position_ = 0;
    };
    void insert_position(int pos_vector)
    { insert_position_ += pos_vector;
      if (insert_position_ > length_)
        insert_position_ = length_;
      if (insert_position_ < 0)
        insert_position_ = 0;
    };
    int start_position() { return starting_position_; };
    int insert_position() { return insert_position_; };
    void reset_start() { starting_position_ = 0; };
    void reset_insert() { insert_position_ = 0; };
    char *shifted_value();
    char *value() { return string_; };
    int shifted_len();
    void insert_char(int c, int pos = -1);
    void delete_char(int pos = -1);
    int length() { return length_; };
    Table_String &operator = (const char *new_value);
    void set(const char *new_value);
    int current_char();
    int previous_char();
}
```

#### DESCRIPTION

This is a string class used by WTable which provides functionality for keeping track of an insertion point and starting point. It is used for values in a text edit window.

## MEMBER FUNCTIONS

Table\_String::Table\_String(char \*value)

Description: Constructor accepting a value. returns: void

Table\_String::Table\_String(Table\_String& value)

Description: Copy Constructor. returns: void

Table\_String::~~Table\_String()

Description: Destructor. Deletes value. returns: void

char \*Table\_String::shifted\_value()

Description: Return char value starting at starting position. returns: char \*, shifted value.

int Table\_String::shifted\_len()

Description: Return length of string starting at starting position. returns: int, shifted length.

void Table\_String::insert\_char(int c, int pos)

Description: Insert character at insert position or arg. returns: void

void Table\_String::delete\_char(int pos )

Description: Delete character at insert position or arg. returns: void

Table\_String &Table\_String::operator = (const char \*new\_value)

Description: Sets the value to new\_value. returns: Table\_String &, \*this.

void Table\_String::set(const char \*new\_value)

Description: Sets the value to new\_value(really). returns: void

Windui(5) Last change: Wed Jan 5 18:02:40 1994 2

Table\_String(5) Gatec Manual Table\_String(5)

int Table\_String::current\_char()

Description: Accessor function. returns: int, char at insert\_position, or 0 if at end.

int Table\_String::previous\_char()

Description: Accessor function. returns: int, char before insert position or 0 if at beginning.



## FILES

tblstrng.c tblstrng.h

## 2.5.6.8 WTButton

### NAME

WTButton - defines a Windows pushbutton used by the DUI Windows client.

### SYNOPSIS

```
#include "WTButton.H"
```

```
class WTButton : public TButton { protected:
    w_Command * dui_element_;
    static HFONT resource_font_;
    Pushbutton_Bitmap *bitmap_;
public:
    int end_view_;
    WTButton(PWindowsObject aParent, int aId, int aX, int aY, int aW, int aH, LPSTR
aTitle, w_Command * aSibling, PTModule aModule = NULL, int aEnd = 0, BOOL isDefault =
0);
    ~WTButton();
    w_Command * dui_element() { return dui_element_; };
    virtual void WMCommand(RTMessage Msg) = [WM_FIRST + WM_COMMAND];
    /* virtual void WMPaint(RTMessage Msg) = [WM_FIRST + WM_PAINT];
    */ virtual int end_view() { return end_view_; };
    virtual int end_view(int aEnd) { end_view_ = aEnd; return 1; };
    virtual void SetupWindow();
    virtual void ODADrawEntire(DRAWITEMSTRUCT &DrawInfo);
    virtual void ODASelect(DRAWITEMSTRUCT &DrawInfo);
    virtual void ODAFocus(DRAWITEMSTRUCT &DrawInfo);
    void iupdate();
    void DuiSetup();
}
```

### DESCRIPTION

This class is a derivative of the Borland TButton with DUI functionality added. It is used as the Windows element for the DUI\_Command and DUI\_End\_Command.

Functionality is also added to allow buttons to be represented as bitmaps set according to the buttons resources (see Windui).

### MEMBER FUNCTIONS

aY, int aW, WTButton::WTButton(PWindowsObject aParent, int

aId, int aX, int                    int aH, LPSTR aTitle, w\_Command \*  
aSibling,                    PModule aModule , int aEnd , BOOL isDefault)

Description: Constructor for the 32-bit TButton object. It checks resources to find out if there is a bitmap associated with this command and what font commands will have and sets them if need be. returns: void

WTButton::~~WTButton()

Description: Destructor for 32-bit TButton object. deletes bitmap if present. returns: void.

void WTButton::WMCommand(RTMessage Msg)

Description: This function is called when the user clicks this button. The button will tell it's 32-bit command sister to choose herself, then it tells the view to send itself back to the application. If it is an End\_Command type it asks the parent window to destroy itself. Responds to WM\_COMMAND messages which are relayed by the parent window(which is the one who really gets the message). returns: void.

void WTButton::SetupWindow()

Description: This function is called when the window receives a WM\_CREATE message. It is redefined here to compute the button size. The buttons height is the height of it's label in current font or "height" if specified. The buttons width is the width of it's label in font or "length" of upper-case characters in font, or "height" if specified. It calls TButton::SetupWindow() first to setup the other attributes of TButton(), it then resizes itself dependent on the resources it has. It actually calls DuiSetup() which does most of this. returns: void.

WTButton::DuiSetup()

Description: Determines and sets the buttons height, width and font. returns: void.

void WTButton::iupdate()

Description: Calls DuiSetup if the buttons name has changed else just checks the buttons read only status and disables it if read only is true. returns: void

void     WTButton::ODADrawEntire(DRAWITEMSTRUCT  
&DrawInfo)

Description: Is called when the button needs to be drawn and it has a bitmap associated with it. It paints the bitmap according to the button's selection status. returns: void

void WTButton::ODAFocus(DRAWITEMSTRUCT &DrawInfo)

Description: Is called when the button gets or loses the focus. It draws the button accordingly. returns:  
void

void WTButton::ODASelect(DRAWITEMSTRUCT &DrawInfo)  
Description: Is called when the button is pushed and when it is released. It draws the button in one of those states. returns: void

## FILES

wtbutton.cpp wtbutton.hpp

## 2.5.6.9 WTCheckBox

### NAME

WTCheckBox - defines a Windows check box used by the DUI Windows client.

### SYNOPSIS

```
#include "WTCheckBox.H"
```

```
class WTCheckBox : public TCheckBox { protected:  
    w_Toggle * dui_element_  
    static HFONT resource_font_  
public:  
    WTCheckBox(PTWindowsObject aParent, int aId, int aX, int aY, int aW, int aH, LPSTR  
aTitle, w_Toggle * aSibling, PTGroupBox aGroup = NULL, PTModule aModule = NULL);  
    ~WTCheckBox();  
    w_Toggle * dui_element() { return dui_element_; };  
    virtual void WMCommand(RTMessage Msg) = [WM_FIRST + WM_COMMAND];  
    virtual void SetupWindow();  
    void DuiSetup();  
    void iupdate();  
}
```

### DESCRIPTION

This class is a derivative of Borland's TCheckBox with dui functionality added. It is the Window's element for the DUI\_Toggle(which see) class.

### MEMBER FUNCTIONS

int aY, WTCheckBox::WTCheckBox(PTWindowsObject aParent,  
int aId, int aX, int aW, int aH, LPSTR aTitle, w\_Toggle \*  
aSibling, PTGroupBox aGroup, PTModule aModule)  
Description: Constructor for the dui TCheckBox object. It sets the  
font for WTCheckBox if provided in the resources (see Windui).  
returns: void

WTCheckBox::~~WTCheckBox()

Description: Destructor for dui TCheckBox object. returns: void

void WCheckBox::WMCommand(RTMessage Msg)

Description: This function is called when the user clicks this button. The button will tell it's dui toggle sister to toggle herself. returns: void

void WCheckBox::SetupWindow()

Description: This function is called when the window receives a WM\_CREATE message. It is redefined here to fill in values from the Toggle dui element. It calls TCheckBox::SetupWindow() first to setup the other attributes of TCheckBox. The function DuiSetup() actually does most of the work. returns: void

WCheckBox::DuiSetup()

Description: This function sets the height, width and font for the check box. returns: void

void WCheckBox::iupdate()

Description: This function is called when the widget is received from the application. If the name of the widget has changed it calls DuiSetup() else it just sets the check status. returns: void

## FILES

wtcheckb.cpp wtcheckb.hpp

## 2.5.6.10 WTComboBox

### NAME

WTComboBox - defines a Windows' Combo Box used by the DUI Windows' client.

### SYNOPSIS

```
#include "WTComboBox.H"
```

```
class WTComboBox: public TComboBox { protected:
w_Selection *dui_element_;
static HFONT resource_font_;
WTStatic *Title_;
public:
WTComboBox(PTWindowsObject aParent, int aId, int aX, int aY, int aW, int aH, w_Selection
* aSibling, PTModule aModule = NULL, DWORD aStyle = CBS_DROPDOWNLIST);
~WTComboBox();
w_Selection *dui_element() { return dui_element_; };
TStatic *Title() { return Title_; };
virtual void WMCommand(RTMessage Msg) = [WM_FIRST + WM_COMMAND];
virtual void SetupWindow();
void DuiSetup();
void iupdate();
}
```

### DESCRIPTION

This class is a derivative of Borland's TComboBox with DUI functionality added. It is a Windows element for the DUI\_Selection(which see) class. It is one of the optional representations for this class. (See DUI).

### MEMBER FUNCTIONS

WTComboBox::WTComboBox(PTWindowsObject aParent, int aId, int aX, int aY, int aW, int aH, w\_Selection \* aSibling, PTModule aModule, DWORD aStyle)

Description: Constructor for dui TComboBox control. It establishes the font for WTComboBox if necessary. returns: void

WTComboBox::~~WTComboBox()

Description: Destructor for dui TComboBox control. returns: void

void WTComboBox::WMCommand(RTMessage Msg)

Description: This function is called when the user selects a new string in the ComboBox. When this happens we want to select it in the dui Selection object as well. returns: void

void WTComboBox::SetupWindow()

Description: This function is called when the window receives a WM\_CREATE message. It is redefined here to fill in values from the selection dui element. It calls TComboBox::SetupWindow() first to setup the other attributes of TComboBox(), it then calls DuiSetup() to set up this instance. returns: void

WtComboBox::DuiSetup()

Description: Sets up the height, width, font and initial value for this widget. returns: void

void WtComboBox::iupdate()

Description: This function is called when the associated DUI object is received from the application. If the name is different it calls DuiSetup() otherwise it just resets the contents of the Windows' element. returns: void

## FILES

wtcombob.cpp wtcombob.hpp

### 2.5.6.11 WTEdit

#### NAME

WTEdit - defines a Windows' edit used by the DUI Windows' client.

#### SYNOPSIS

```
#include "WTEdit.H"
```

```
class WTEdit: public TEdit { protected:
w_Field *dui_element_;
static HFONT resource_font_;
WTStatic *Title_;
public:
WTEdit(PtWindowsObject aParent, int aId, LPSTR aContents, int aX, int aY, int aW, int aH,
w_Field * aSibling, PTModule aModule = NULL, int aMult = 0);
~WTEdit();
w_Field *dui_element() { return dui_element_; };
TStatic *Title() { return Title_; };
virtual void WMCommand(RTMessage Msg) = [WM_FIRST + WM_COMMAND];
virtual void SetupWindow();
void DuiSetup();
void iupdate();
}
```

#### DESCRIPTION

This class is a derivative of Borland's TEdit with DUI functionality added. It acts as the Windows' element for the DUI\_Field(which see) class. (See Windui).

#### MEMBER FUNCTIONS

int aX, WTEdit::WTEdit(PtWindowsObject aParent, int aId, LPSTR aContents, int aY, int aW, int aH, w\_Field \* aSibling, PTModule aModule, int aMult) Description:

WTEdit(): constructor for dui TEdit control. Sets up font for WTEdit if necessary. returns: void

WTEdit::~~WTEdit()

Description: Destructor for dui TEdit control. returns: void

void WTEdit::WMCommand(RTMessage Msg)

Description: This function is called when the edit control receives an EN\_KILLFOCUS message from windows. We want to run the contents of the field against it's modifiers and constraints when the user attempts to move out of this field, and give him an error notification when the constraints fail. This may be a nazi way of applying field constraints. But we want real constraints. If the

value is invalid a message box is displayed with the error in it, The user is given the choice of continuing using the old value(the value before he entered anything) or retrying. returns: void

`void WTEdit::SetupWindow()`

Description: This function is called when the window receives a WM\_CREATE message. It is redefined here to fill in values from the field dui element. It calls TEdit::SetupWindow() first to setup the other attributes of TEdit(), it then fills in the value, and resizes itself dependent on the resources it has. It calls DuiSetup() to do the sizing. returns: void

`WTEdit::DuiSetup()`

Description: Sets the height, width, font and initial values for list box. returns: void

`void WTEdit::iupdate()`

Description: This function is called when the associated widget is received from the application. If the name has changed it calls DuiSetup() otherwise it resets the value of the edit. returns: void

## FILES

wteedit.cpp wteedit.hpp



## 2.5.6.12 WTGroupBox

### NAME

WTGroupBox - defines a Windows' group box used by the DUI Windows' client.

### SYNOPSIS

```
#include "WTGroupBox.H"
```

```
class WTGroupBox: public TGroupBox { protected:
w_Component *dui_element_;
static HFONT resource_font_;
public:
WTGroupBox(PWindowsObject aParent, int aId, LPSTR aContents, int aX, int aY, int aW,
int aH, w_Component * aSibling, PTModule aModule = NULL);
~WTGroupBox();
w_Component *dui_element() { return dui_element_; };
virtual void SetupWindow();
void iupdate();
}
```

### DESCRIPTION

This class is a derivative of Borland's TGroupBox with DUI functionality added. It acts as the Windows' element for the DUI\_Group(which see) class. (See Windui). It is either not visible at all or visible as a box around the components in its group.

### MEMBER FUNCTIONS

aContents, WTGroupBox::WTGroupBox(PWindowsObject  
aParent, int aId, LPSTR int aX,int aY, int aW, int  
aH, w\_Component \* aSibling, PTModule aModule)  
Description: Constructor for dui TGroupBox control. Establishes  
font if necessary. returns: void

WTGroupBox::~~WTGroupBox()

Description: Destructor for dui TGroupBox control. returns: void

void WTGroupBox::SetupWindow()

Description: Calls TGroupBox::SetupWindow() and sets the font  
on initial startup. returns: void

void WTGroupBox::iupdate()

Description: Sets a new name if name has changed. returns: void

### FILES

wtgroupb.cpp wtgroupb.hpp

### 2.5.6.13 WTLListBox

#### NAME

WTLListBox - defines a Windows' list box used by the DUI Windows' client.

#### SYNOPSIS

```
#include "WTLListBox.H"
```

```
class WTLListBox: public TListBox { protected:
w_Selection *dui_element_;
static HFONT resource_font_;
WTStatic *Title_;
public:
WTLListBox(PTWindowsObject aParent, int aId, int aX, int aY, int aW, int aH, w_Selection *
aSibling, PTModule aModule = NULL, int multi = 0);
~WTLListBox();
w_Selection *dui_element() { return dui_element_; };
TStatic *Title() { return Title_; };
virtual void WMCommand(RTMessage Msg) = [WM_FIRST + WM_COMMAND];
virtual void WMChar(RTMessage Msg) = [WM_FIRST + WM_CHAR];
virtual void SetupWindow();
void DuiSetup();
void iupdate();
void selectitems();
int itemisselected(int idx);
}
```

#### DESCRIPTION

This class is a derivative of Borland's TListBox with DUI functionality added. It acts as the Windows' element for the DUI\_Selection and DUI\_Multi\_Selection(which see) class. (See Windui).

#### MEMBER FUNCTIONS

WTLListBox::WTLListBox(PTWindowsObject aParent, int aId, int aX, int aY, int aW, int aH, w\_Selection \* aSibling, PTModule aModule, int multi)

Description: Constructor for dui TListBox control. returns: void

WTLListBox::~~WTLListBox()

Description: Destructor for dui TListBox control. returns: void

void WTLListBox::WMCommand(RTMessage Msg)

Description: This function is called when the user selects a new

string in the ListBox. When this happens we want to select it in the dui Selection object as well. For Multi\_Selections it is called for deselection as well. returns: void

void WTLListBox::WMChar(RTMessage Msg)

Description: This function responds to the WM\_CHAR message. It intercepts 'A' or 'a' keys and selects/deselects all items in list. returns: void

void WTLListBox::SetupWindow()

Description: This function is called when the window receives a WM\_CREATE message. It is redefined here to fill in values from the selection dui element. It calls TListBox::SetupWindow() first to setup the other attributes of TListBox() it then calls DuiSetup() to set up this instance. returns: void

WTLListBox::DuiSetup()

Description: Sets the height, width, font and initial values of this list box. returns: void

void WTLListBox::iupdate()

Description: This function is called when the associated dui element is received from the application. If the name as changed it calls DuiSetup() otherwise it just resets the contents of the list box. returns:  
void

void WTLListBox::selectitems()

Description: This function sets the selected items in the Windows list box according to those selected in the dui element. returns: void

int WTLListBox::itemisselected(int idx)

Description: This function determines if the indexed item is selected in the Windows' list box. returns:  
int, TRUE if selected, FALSE otherwise.

## FILES

wtlistbo.cpp wtlistbo.hpp

## 2.5.6.14 WTRadioButton

### NAME

WTRadioButton - defines a Windows' radio button used by the  
DUI Windows' client.

### SYNOPSIS

```
#include "WTRadioButton.H"
```

```
class WTRadioButton : public TRadioButton { protected:  
    w_Toggle * dui_element_  
    static HFONT resource_font_  
public:  
    WTRadioButton(PTWindowsObject aParent, int aId, int aX, int aY, int aW, int aH, LPSTR  
aTitle, w_Toggle * aSibling, PTGroupBox aGroup = NULL, PTModule aModule = NULL);  
    ~WTRadioButton();  
    w_Toggle * dui_element() { return dui_element_; };  
    virtual void WMCommand(RTMessage Msg) = [WM_FIRST + WM_COMMAND];  
    virtual void SetupWindow();  
    void DuiSetup();  
    void iupdate();  
}
```

### DESCRIPTION

This class is a derivative of Borland's TRadioButton with DUI  
functionality added. It is one of the options for representation of  
the DUI\_Toggle(which see) class, the other is  
WTCheckBox(which see). (See Windui).

### MEMBER FUNCTIONS

```
int aX, int  
WTRadioButton::WTRadioButton(PTWindowsObject aParent,  
int aId, int aW,int aH, LPSTR aTitle, w_Toggle *  
aSibling, PTGroupBox aGroup, PTModule aModule)  
Description: Constructor for the dui TRadioButton object. returns:  
void
```

```
WTRadioButton::~WTRadioButton()  
Description: Destructor for dui TRadioButton object. returns: void
```

```
void WTRadioButton::WMCommand(RTMessage Msg)  
Description: This function is called when the user clicks this  
button. The button will tell it's dui toggle sister to toggle herself.  
returns: void
```

```
void WTRadioButton::SetupWindow()  
Description: This function is called when the window receives a
```

WM\_CREATE message. It is redefined here to fill in values from the Toggle dui element. It calls TRadioButton::SetupWindow() first to setup the other attributes of TRadioButton. It then calls DuiSetup() to set up this instance. returns: void

WTRadioButton::DuiSetup()

Description: Sets the height, width, font and initial value of the radio button. returns: void

void WTRadioButton::iupdate()

Description: This function is called when the associated dui element is received from the application. If the name has changed it calls DuiSetup() otherwise it just sets the value fo the radio button. returns: void

## FILES

wtradiob.cpp wtradiob.hpp

### 2.5.6.15 WTStatic

#### NAME

WTStatic - defines a Windows' static used by the DUI Windows' client.

#### SYNOPSIS

```
#include "WTStatic.H"
```

```
class WTStatic: public TStatic { protected:
w_Label *dui_element_;
static HFONT resource_font_;
Pushbutton_Bitmap *bitmap_;
public:
WTStatic(PTWindowsObject aParent, int aId, LPSTR aTitle, int aX, int aY, int aW, int aH,
WORD aLen, w_Label * aSibling, PTModule aModule = NULL, int is_title = 0);
~WTStatic();
w_Label *dui_element() { return dui_element_; };
virtual void SetupWindow();
void iupdate();
void DuiSetup();
void SetResourceFont(HFONT newfont);
virtual void WMShowWindow(RTMessage Msg) = [WM_FIRST + WM_SHOWWINDOW];
virtual void WMPaint(RTMessage Msg) = [WM_FIRST + WM_PAINT];
}
```

#### DESCRIPTION

This class is a derivative of Borland's TStatic with DUI functionality added. It acts as the Windows' element for the DUI\_Label(which see) class. (See Windui).

#### MEMBER FUNCTIONS

tle, int aX, WTStatic::WTStatic(PTWindowsObject aParent, int aId, LPSTR aTitle, int aY, int aW, int aH, WORD aLen, w\_Label \* aSibling, PTModule aModule, int is\_title)

Description: constructor for dui TStatic control. returns: void

WTStatic::~~WTStatic()

Description: Destructor for dui TStatic control. returns: void

void WTStatic::SetupWindow()

Description: This function is called when the window receives a WM\_CREATE message. It is redefined here to fill in values from the label dui element. It calls TStatic::SetupWindow() first to setup the other attributes of TStatic, it then resizes itself dependent on the resources it has. returns: void

WTStatic::DuiSetup()

Description: Sets height, width and font. returns:  
void

void WTStatic::iupdate()

Description: This function is called when the associated dui element is received from the application. It resets the title if it is different. returns: void

## FILES

wtstatic.cpp wtstatic.hpp

## 2.5.6.16 WText

### NAME

WText - defines a Windows' edit used by the DUI Windows' client.

### SYNOPSIS

```
#include "WText.H"
```

```
class WText: public TEdit { protected:
w_Text *dui_element_;
static HFONT resource_font_;
WStatic *Title_;
public:
WText(PWindowsObject aParent, int aId, LPSTR aContents, int aX, int aY, int aW, int aH,
w_Text * aSibling, PTModule aModule = NULL);
~WText();
w_Text *dui_element() { return dui_element_; };
TStatic *Title() { return Title_; };
virtual void WMCommand(RTMessage Msg) = [WM_FIRST + WM_COMMAND];
virtual void WMChar(RTMessage Msg) = [WM_FIRST + WM_CHAR];
virtual void SetupWindow();
void DuiSetup();
void iupdate();
}
```

### DESCRIPTION

This class is a derivative of Borland's TEdit with DUI functionality added. It acts as the Windows' element for the DUI\_Text(which see) class. (See Windui).

### MEMBER FUNCTIONS

int aX, WText::WText(PWindowsObject aParent, int aId, LPSTR aContents, int aY, int aW, int aH, w\_Text \* aSibling, PTModule aModule ) Description: Constructor for dui TEdit control. returns: void

WText::~~WText()

Description: Destructor for dui TEdit control. returns: void

void WText::WMCommand(RTMessage Msg)

Description: ENKillFocus() This function is called when the edit control receives an EN\_KILLFOCUS message from windows. It sets the value of the DUI\_Text according to what was entered by the user. returns: void

void WText::SetupWindow()



Description: This function is called when the window receives a WM\_CREATE message. It is redefined here to fill in values from the text dui element. It calls TEdit::SetupWindow() first to setup the other attributes of TEdit(), it then fills in the value, and resizes itself dependent on the resources it has by calling DuiSetup().  
returns: void

WText::DuiSetup()

Description: This function sets the height, width, font and initial value of the edit. returns: void

void WText::iupdate()

Description: This function is called when the associated dui element is received from the application. If the name changes it calls DuiSetup() else it just resets the contents of the edit. returns: void

## FILES

wtext.cpp wtext.hpp

### 2.5.6.17 WTWindow

#### NAME

WTWindow - defines a Windows' window used by the DUI Windows' client.

#### SYNOPSIS

```
#include "WTWindow.H"
```

```
class WTWindow : public TWindow { protected:
    w_View * dui_element_;
    HFONT resource_font_;
public:
    WTWindow(PAllWindowsObject aParent, LPSTR aTitle,      w_View * aSibling, PTModule
aModule = NULL);
    ~WTWindow();
    w_View * dui_element() { return dui_element_; };
    virtual void WMCommand(RTMessage Msg) = [WM_FIRST + WM_COMMAND];
    virtual void WMSetCursor(RTMessage Msg) = [WM_FIRST + WM_SETCURSOR];
        virtual void WMShowWindow(RTMessage Msg)      =      [WM_FIRST      +
WM_SHOWWINDOW];
    virtual void SetupWindow();
    virtual void iupdate();
    virtual void DuiSetup();
}
```

#### DESCRIPTION

This class is a derivative of Borland's TWindow with DUI functionality added. It acts as the Windows' element for the W\_View(which see) class. (See Windui).

#### MEMBER FUNCTIONS

WTWindow::WTWindow(PAllWindowsObject aParent, LPSTR aTitle, w\_View \* aSibling, PTModule aModule )

Description: Constructor for the dui TWindow object. It is a scrolling window, so the style is set and the Scroller pointer is set. We also want keyboard handling for our views. returns: void

WTWindow::~~WTWindow()

Description: Destructor for dui TWindow object. returns: void

void WTWindow::WMCommand(RTMessage Msg)

Description: This function handles all WM\_COMMAND messages. The commands the window responds to are:

SEND\_YOURSELF - When a w\_Command widget is selected the w\_View needs to send itself back to the application. This message is sent by WTButton an interface sibling to w\_Command. All other

messages are returned to the window that is referenced by the message because all the other widgets respond to their own WM\_COMMAND messages. returns: void

void WTWindow::WMSetCursor(RTMessage Msg)

Description: This function is called when the window receives a WM\_SETCURSOR message. It calls the applications main window cycle\_cursor function. returns: void

void WTWindow::WMShowWindow(RTMessage Msg)

Description: This function is called when the window receives a WM\_SHOWWINDOW message. It releases the capture of the mouse if captured and restores the wait mode of the application before it calls DefWndProc. returns: void

void WTWindow::SetupWindow()

Description: This function is called when the window receives a WM\_CREATE message. It calls SetupWindow for its base class TWindow. returns: void

WTWindow::DuiSetup()

Description: This function determines the size of this window and removes scrolling on windows that can fit on the screen. It then moves the window to the top left corner and the appropriate size. If the window is a dialog it centers it. returns: void

WTWindow::iupdate()

Description: This function is called when the associated dui element is received from the application. It resets the title if the name has changed and calls DuiSetup(). returns: void

## FILES

wtwindow.cpp wtwindow.hpp

## 2.5.6.18 WTable

### NAME

WTable - Windows implementation of a DUI\_Table.

### SYNOPSIS

```
#include "WTable.H"
```

```
class WTable: public TListBox { private:
    Table_String **column_buffers;
    RECT text_rect_;
    RECT innerframe_rect_;
    RECT outerframe_rect_;
    w_Table *dui_element_;
    static HFONT resource_font_;
    WStatic *Title_;
    int width_unit_;
    int current_selected_idx_;
    int current_column_idx_;
    int override_selection_;
    int kill_deselect_;
    int cannot_cancel_;
public:
    WTable(PWindowsObject aParent, int aId, int aX, int aY, int aW, int aH, w_Table * aSibling,
    PTModule aModule = NULL);
    ~WTable();
    w_Table *dui_element() { return dui_element_; };
    TStatic *Title() { return Title_; };
    virtual void SetupWindow();
    void DuiSetup();
    void iupdate();
    void set_caret(int col);
    int increment_caret(HDC hDC, int c);
    int decrement_caret(HDC hDC, int c);
    const char * save_current_row();
    int save_with_dialog();
    RECT *selected_col_text_rect();
    RECT *outerframe_rect(int col, int row, RECT *lprect = 0);
    RECT *innerframe_rect(int col, int row, RECT *lprect = 0);
    RECT *text_rect(int col, int row, RECT *lprect = 0);
    void frame_cols(DRAWITEMSTRUCT &DrawInfo, int is_selected);
    void shift_display_text(HDC hDC, int col, int row, int shift_vector = 0, int is_selected =
0);
    void reset_itemData();
    void clear_buffers();
    void reset_buffers(int idx);
    virtual void WMChar(RTMessage Msg) = [WM_FIRST + WM_CHAR];
    virtual void WMDlgCode(RTMessage Msg) = [WM_FIRST + WM_GETDLGCODE];
```

```

virtual void WMSetCursor(RTMessage Msg) = [WM_FIRST + WM_SETCURSOR];
virtual void WMCommand(RTMessage Msg) = [WM_FIRST + WM_COMMAND];
virtual void WMSetFocus(RTMessage Msg) = [WM_FIRST + WM_SETFOCUS];
virtual void WMKillFocus(RTMessage Msg) = [WM_FIRST + WM_KILLFOCUS];
virtual void WMKeyUp(RTMessage Msg) = [WM_FIRST + WM_KEYUP];
virtual void WMKeyDown(RTMessage Msg) = [WM_FIRST + WM_KEYDOWN];
virtual void WMVScroll(RTMessage Msg) = [WM_FIRST + WM_VSCROLL];
    virtual void WMLButtonDown(RTMessage Msg) = [WM_FIRST
WM_LBUTTONDOWN];
virtual void ODADrawEntire(DRAWITEMSTRUCT &DrawInfo);
virtual void ODASelect(DRAWITEMSTRUCT &DrawInfo);
virtual void ODAFocus(DRAWITEMSTRUCT &DrawInfo);
}

```

## DESCRIPTION

This class defines the Windows object that fulfills the display behavior for a `DUI_Table`(which see) widget. It is a list box with editable columns. So an entry can be selected and then edited by column. New rows can be inserted or deleted as well. When widget has the focus, the active keys are:

Mouse click - select a row. up and down arrow keys - select a row.  
insert key - insert a new blank row before the current row. delete  
key - delete the current row. left and right arrow keys - move left  
and right one character in the currently selected column of the  
current row. tab key - move to the next column in current row.  
shift-tab key - move to the previous column of the current row.  
backspace - delete char backwards.

## MEMBER FUNCTIONS

`WTable::WTable(PTWindowsObject aParent, int aId, int aX, int  
aY, int aW, int aH, w_Table * aSibling, PTModule aModule)`  
Description: Constructor for dui WTable control. This control is a  
listbox whose contents are editable. The listbox has the ownerdraw  
style so it handles drawing the table contents as well as editing the  
table contents. returns: void

`WTable::~~WTable()`

Description: Destructor for WTable. It resets no\_ielement and  
deletes allocated memory. returns:  
void

`void WTable::WMChar(RTMessage Msg)`

Description: This function responds to a `WM_CHAR` message for  
Windows. It intercepts characters typed and inserts them into the  
current column and row, if there is one and it is not read only.  
returns: void

`void WTable::WMDlgCode(RTMessage Msg)`

Description: Responds to a `WM_DLGCODE` message, setting the

kind of keyboard input that we want. returns: void

`void WTable::WMCommand(RTMessage Msg)`

Description: Responds to WM\_COMMAND message with a parameter of CAPTURE\_THE\_FOCUS. Whereupon it sets the focus to the current window. It will receive this message from WMKillFocus() if there was an error in the last entered text. returns: void

`void WTable::WMKeyUp(RTMessage Msg)`

Description: Responds to a WM\_KEYUP message and does nothing. returns: void

`void WTable::WMKeyDown(RTMessage Msg)`

Description: Responds to a WM\_KEYDOWN message, and traps some special keys to do special processing:

DELETE - delete current row and select next row; ESCAPE - if in the middle of editing? put back old contents of row but leave row selected. INSERT - add new row above current row and select. RIGHT-ARROW - goto next letter in current column stopping at last letter LEFT-ARROW - goto previous letter in current column stopping at first letter TAB - goto next column if last column call default. SHIFT-TAB - goto previous column if first column call default. BACKSPACE - delete char back-wards. returns: void

`void WTable::WMVScroll(RTMessage Msg)`

Description: Responds to WM\_VSCROLL message. It checks to make sure caret is visible if it should be and hidden if it should be. returns: void

`void WTable::WMLButtonDown(RTMessage Msg)`

Description: Responds to WM\_LBUTTONDOWN message. Selects appropriate row. returns: void

`void WTable::WMSetFocus(RTMessage Msg)`

Description: Responds to WM\_SETFOCUS message by creating the caret. returns: void

`void WTable::WMKillFocus(RTMessage Msg)`

Description: Responds to WM\_KILLFOCUS by checking the validity of the last entered text and destroying the caret. returns: void

`void WTable::WMSetCursor(RTMessage Msg)`

Description: If the cursor is on the currently selected items set it to I-beam cursor. Responds to WM\_SETCURSOR message. returns: void

`void WTable::SetupWindow()`

Description: This function is called when the window receives a

WM\_CREATE message. It is redefined here to fill in values from the selection dui element. It calls TListBox::SetupWindow() first to setup the other attributes of TListBox(), then it calls DUISetup(). returns: void

WTable::DuiSetup()

Description: This function is called when a window's element is first created and when the name of the widget changes. It reads resources establishing the width, height and font of the new or changed widget. It also resets the rows in the window's element and creates new edit buffers for each column. returns: void

void WTable::iupdate()

Description: This function is called when an update to this widget is sent from the application. It determines if the name of the widget has changed and calls DuiSetup() if it has(because it can not be sure that widget will remain the same size), otherwise it just resets the rows displayed in the windows' element. returns: void

void WTable::ODADrawEntire(DRAWITEMSTRUCT &DrawInfo)

Description: This function responds to a WM\_DRAWITEM message whereupon it draws the framing around the columns and row highlighting the row if it is selected. The message is sent for each item in the list box. returns: void

void WTable::ODASelect(DRAWITEMSTRUCT &DrawInfo)

Description: This function is called when an item in the list box has changed its selection status (selected or deselected). When an item is selected the contents of the previously selected row is saved, and if the save failed that row is selected again nullifying the current selection activity, otherwise it proceeds. If the row has been deselected it proceeds as well. In both cases the item is redrawn reflecting its new selection status. returns: void

void WTable::ODAFocus(DRAWITEMSTRUCT &DrawInfo)

Description: This function is called for each item when the focus is set on the WTable object. It calls the ODAFocus() of its parent class TListBox. returns: void

is\_selected)

void WTable::frame\_cols(DRAWITEMSTRUCT &DrawInfo, int Description: This function actually does the drawing of the borders for each column in the desired rows as well as displays the text that that row contains by calling shift\_display\_text(). It does selection drawing if the item is selected. returns: void

RECT \* WTable::selected\_col\_text\_rect()

Description: This function returns the RECT structure which

describes the rectangle which encloses the selected text. returns: RECT \*, the selected text RECT structure.

RECT \* WTable::text\_rect(int col, int row, RECT \*lprect)

Description: This function returns the RECT structure that describes the rectangle enclosing the text of the passed column and row. returns: RECT \*, the text rectangle.

RECT \* WTable::innerframe\_rect(int col, int row, RECT \*lprect)

Description: This function returns the RECT structure that encloses the innerframe of the passed column and row. returns: RECT \*, the innerframe of the column.

RECT \* WTable::outerframe\_rect(int col, int row, RECT \*lprect)

Description: This function returns the RECT structure that encloses the outer frame of the passed column and row. If the lprect arg is 0, the rectangle is retrieved through Windows'. returns: RECT \*, the outer frame of the column.

shift\_vector, void WTable::shift\_display\_text(HDC hDC, int col, int row, int int is\_selected) Description: This function displays the text for desired row and column. If the column is in the selected row, the text may have been shifted by cursor movement and editing so the shift vector is used to display as much of the string as is currently visible. returns: void

void WTable::set\_caret(int col)

Description: This function sets the caret(the blinking edit cursor) to the beginning of the desired row and resets the shifted buffer to the beginning. returns: void

int WTable::increment\_caret(HDC hDC, int c)

Description: This function moves the caret to the right by the width of the passed character in the current font. returns: int, -1 if at end of string, 0 if at end of rectangle, 1 otherwise.

int WTable::decrement\_caret(HDC hDC, int c)

Description: This function moves the caret to the left by the width of the passed character in the current font. Stopping at the borders of the rectangle. returns: int, -1 if at beginning of string, 0 if at beginning of rectangle, 1 otherwise.

void WTable::reset\_itemData()

Description: This function resets the data attached to the list box items to be in sync with the contents of the DUI Widget. returns: void

const char \* WTable::save\_current\_row()

Description: This function saves the data entered by the user into



the DUI Widget and checks to make sure there were no errors.  
returns: char \*, the error message if error, else 0.

int WTable::save\_with\_dialog()

Description: This function calls save\_current\_row() and if there were errors pops up a message box relaying the error message.  
returns: int, 1 if error, 0 otherwise.

void WTable::clear\_buffers()

Description: Sets contents of the edit buffers used to capture user data to "". returns: void

Description:

sets buffers back to their original contents from the DUI Widget.  
returns: void

## FILES

wtable.cpp wtable.hpp

### 2.5.6.19 w\_Command

#### NAME

w\_Command - Windui extensions to DUI\_Command.

#### SYNOPSIS

```
#include "w_Command.H"

protected:
class WtButton * interface_element_;
class WtGroupBox * group_box_;
w_Component ** old_components_;
int old_component_count_;
public:
WtButton * interface_element() { return interface_element_; };
virtual int icreate(w_View *aParent);
virtual int reposition(int X, int Y);
virtual int resize(int W, int H);
virtual int isize(int *W, int *H);
virtual boolean read_only() const { return w_Component::read_only(); };
virtual void client_construct();
virtual void client_destruct();
    virtual void hide_unused_components();
    virtual void store_components();
const char * check_view_invalid();
void no_ielement() { interface_element_ = 0;
hidden_or_shown_ = 0;};
void no_group() { group_box_ = 0; };
void hide_component();
void show_component();
void make_window();
void hide_old();
virtual void set_hide_show(int hsarg = -1);
void really_show();
void really_show_old();
void receive();
```

#### DESCRIPTION

These methods are extensions to the DUI\_Command(which see) class defined in the DUI Toolkit. They provide additional functionality required by the Windows DUI client (e.g. functions dealing directly with display of this object). It has a pointer to an associated object WtButton(which see) which actually defines the Windows element.

#### MEMBER FUNCTIONS

int w\_Command::icreate(w\_View \*aParent)

Description: This function is called to update an existing command if it has changed. It, in turn, will run through it's list of commands and call their icreate function if necessary. It will also create a new command if this one has not been created yet. returns: int, 0 always.

void w\_Command::really\_show()

Description: This function checks hide\_show\_ and hidden\_or\_shown\_ to see if element must be explicitly displayed or hidden. This function is called on a final pass through a view's instance hierarchy to display or hide the actual elements. The hide\_show\_ flag is set when elements are created or updated. This is done so the view's components can change dynamically. The components always exist once created but may be removed or added to a view at any time during execution therefore components which are no longer part of a view are hidden instead of removed. Components can not be part of two different views at once. Although this is desirable. returns: void

void w\_Command::really\_show\_old()

Description: This function is necessary for objects that act as groups for other objects. w\_Command can have subcommands so it keeps track of its previous set of subcommands so that they can be hidden if no longer a part of this group. Conflicts arising from a component simply switching groups is resolved by using the hide\_show\_ flag which can not be set to hidden once it has been set to shown. returns: void

int w\_Command::reposition(int X, int Y)

Description: This function allows another object to ask this object to reposition itself given the passed coordinates. It operates in two ways, if this is non-group command it just positions its windows element, else it positions a group box surrounding the subcommands and calls reposition on the subcommands after adjusting the coordinates for spacing. returns: int, 1 always

int w\_Command::resize(int W, int H)

Description: This function allows for resizing of the object. returns: int, 1 always.

int w\_Command::isize(int \*W, int \*H)

Description: This function returns its width and height into the arguments passed. returns: int, 1 always.

w\_Command::client\_construct()

Description: This is an addition to the constructor for DUI\_Command. It initializes the data members added in these extensions. returns: void

w\_Command::client\_destruct()

Description: Destructor for client additions. returns:  
void

w\_Command::check\_view\_invalid()

Description: This function calls check\_invalid() for its view's component. Used because the command needs to do it when it is pressed. returns: char \*, the error message or 0.

void w\_Command::store\_components()

Description: This function saves the old component list and count. returns: void

void w\_Command::set\_hide\_show(int hsarg)

Description: Sets the hide\_show\_ flag for this command and all subcommands. returns: void

void w\_Command::hide\_old()

Description: Sets the hide\_show\_ flag to hide for all the old components. returns: void

void w\_Command::receive()

Description: This function is called whenever this object is received from the application. It calls iupdate() on its interface\_element\_ and receive() on all of its subcommands. returns: void

## FILES

w\_comman.cc w\_comman.hh

## 2.5.6.20 w\_Component

### NAME

w\_Component - Windui extensions to DUI\_Component.

### SYNOPSIS

```
#include "w_Component.H"
```

```
private:
protected:
    int hide_show_;
public:
    virtual void client_construct();
    virtual void client_destruct() { return; };
    virtual void hide_unused_components() { return; }
    virtual void store_components() { return; }
    virtual void hide_component() { return; }
    virtual void show_component() { return; }
    virtual void make_window() { return; }
    virtual void hide_old() { hide_show_ = 0; }
    virtual void really_show() { return; };
    virtual void really_show_old() { really_show(); };
    virtual void set_hide_show(int hsarg = -1) { if (hide_show_ == -1 || hsarg == -1) { hide_show_
= hsarg;
} else if (hide_show_ == 0) { hide_show_ = hsarg;
}
}
```

### DESCRIPTION

These methods are extensions to the DUI\_Component(which see) class defined in the DUI Toolkit. They provide additional functionality required by the Windows DUI client.

### MEMBER FUNCTIONS

w\_Component::client\_construct()

Description: Constructor additions for the Windows client.  
returns: void

### FILES

w\_compon.cc w\_compon.hh

## 2.5.6.21 w\_End\_Command

### NAME

w\_End\_Command - Windui extensions to DUI\_End\_Command.

### SYNOPSIS

```
#include "w_End_Command.H"
```

```
public:  
virtual int icreate(w_View *aParent);  
virtual int iupdate();
```

### DESCRIPTION

These methods are extensions to the DUI\_End\_Command(which see) class defined in the DUI Toolkit. They provide additional functionality required by the Windows DUI client (e.g. functions dealing directly with display of this object). It has a pointer to an associated object WtButton(which see) which actually defines the Windows element. This it inherited from DUI\_Command of which it is a derivative.

### MEMBER FUNCTIONS

```
int w_End_Command::icreate(w_View *aParent)
```

Description: This function calls w\_Command::icreate() and sets the end\_view\_ flag to one on its interface\_element\_.  
returns: int, 1 always.

```
int w_End_Command::iupdate()
```

Description: Calls interface\_element\_->iupdate(). returns: void

### FILES

w\_end\_co.cc w\_end\_co.hh

## 2.5.6.22 w\_Field

NAME

w\_Field - Windui extensions to DUI\_Field.

SYNOPSIS

```
#include "w_Field.H"
```

```
/* w_field.HH * Contains definitions specific to w_field as  
modifications for duit * sister class DUI_field. * Kevin Convy 12/16/92  
*/
```

```
protected:
```

```
class WTEdit * interface_element_;
```

```
public:
```

```
WTEdit * interface_element() { return interface_element_; };
```

```
int icreate(w_View *aParent);
```

```
virtual int reposition(int X, int Y);
```

```
virtual int resize(int W, int H);
```

```
virtual int isize(int *W, int *H);
```

```
virtual void client_construct();
```

```
void no_ielement() { interface_element_ = 0;
```

```
hidden_or_shown_ = 0;};
```

```
void hide_component() ;
```

```
void show_component() ;
```

```
void make_window();
```

```
void really_show();
```

```
void receive();
```

## DESCRIPTION

These methods are extensions to the DUI\_Field(which see) class defined in the DUI Toolkit. They provide additional functionality required by the Windows DUI client (e.g. functions dealing directly with display of this object). It has a pointer to an associated object WTEdit(which see) which actually defines the Windows element.

## MEMBER FUNCTIONS

```
int w_Field::icreate(w_View *aParent)
```

Description: This function is called to update an existing field if it has changed. It will also create a new command if this one has not been created yet. returns: void

```
void w_Field::really_show()
```

Description: This function checks hide\_show\_ and hidden\_or\_shown\_ to see if element must be explicitly displayed

or hidden. This function is called on a final pass through a view's instance hierarchy to display or hide the actual elements. The `hide_show_flag` is set when elements are created or updated. This is done so the view's components can change dynamically. The components always exist once created but may be removed or added to a view at any time during execution therefore components which are no longer part of a view are hidden instead of removed. Components can not be part of two different views at once. Although this is desirable. returns: void

`int w_Field::reposition(int X, int Y)`

Description: This function allows another object to ask this object to reposition itself given the passed coordinates. It adjusts the position of its title member and then of its windows element member. returns: int, 1 always

`int w_Field::resize(int W, int H)`

Description: This function allows for resizing of the object. returns: int, 1 always.

`int w_Field::isize(int *W, int *H)`

Description: This function returns its width and height into the arguments passed. returns: int, 1 always.

`w_Field::client_construct()`

Description: This is an addition to the constructor for `DUI_Command`. It initializes the data members added in these extensions. returns: void

`void w_Field::receive()`

Description: This function is called whenever this object is received from the application. It calls `iupdate()` on its `interface_element_`. returns: void

## FILES

`w_field.cc w_field.hh`



### 2.5.6.23 w\_Group

NAME

w\_Group - Windui extensions to DUI\_Group.

SYNOPSIS

```
#include "w_Group.H"
```

w\_Group.HH Contains definitions specific to w\_Group as modifications for duit sister class DUI\_Group.

```
protected:
class WTGroupBox * interface_element_;
w_Component ** old_components_;
int old_component_count_;
public:
WTGroupBox *   interface_element()   {   return interface_element_;
};
int icreate(w_View *aParent);
virtual int reposition(int X, int Y);
virtual int resize(int W, int H);
virtual int isize(int *W, int *H);
virtual void client_construct();
virtual void client_destruct();
virtual void hide_unused_components();
virtual void store_components();
void no_ielement() { interface_element_ = 0;
hidden_or_shown_ = 0;};
void hide_component();
void show_component();
void make_window();
virtual void set_hide_show(int hsarg = -1);
void hide_old();
void really_show();
void really_show_old();
void receive();
```

DESCRIPTION

These methods are extensions to the DUI\_Group(which see) class defined in the DUI Toolkit. They provide additional functionality required by the Windows DUI client (e.g. functions dealing directly with display of this object). It has a pointer to an associated object WTGroupBox(which see) which actually defines the Windows element.

MEMBER FUNCTIONS

`int w_Group::icreate(w_View *aParent)`

Description: This function is called to update an existing group if it has changed. It, in turn, will run through it's list of components and call their icreate function if necessary. It will also create a new group if this one has not been created yet. returns: void

`int w_Group::reposition(int X, int Y)`

Description: This function allows another object to ask this object to reposition itself given the passed coordinates. It adjusts the position its list of components and then itself. returns: int, 1 always

`int w_Group::resize(int W, int H)`

Description: This function allows for resizing of the object. returns: int, 1 always.

`int w_Group::isize(int *W, int *H)`

Description: This function returns its width and height into the arguments passed. returns: int, 1 always.

`w_Group::client_construct()`

Description: This is an addition to the constructor for DUI\_Command. It initializes the data members added in these extensions. returns: void

`w_Group::client_destruct()`

Description: Destructor additions for this object. returns: void

`void w_Group::store_components()`

Description: This function saves the old component list and count. returns: void

`void w_Group::really_show()`

Description: This function checks `hide_show_` and `hidden_or_shown_` to see if element must be explicitly displayed or hidden. This function is called on a final pass through a view's instance hierarchy to display or hide the actual elements. The `hide_show_` flag is set when elements are created or updated. This is done so the view's components can change dynamically. The components always exist once created but may be removed or added to a view at any time during execution therefore components which are no longer part of a view are hidden instead of removed. Components can not be part of two different views at once. Although this is desirable. returns: void

`void w_Group::really_show_old()`

Description: This function is necessary for objects that act as groups for other objects. `w_Group` has a list of components so it must track of its previous set of components so that they can be hidden if no longer a part of this group. Conflicts arising from a component simply switching groups is resolved by using the

hide\_show\_ flag which can not be set to hidden once it has been set to shown. returns: void

void w\_Group::set\_hide\_show(int hsarg)

Description: Sets the hide\_show\_ flag for this group and all of its components. returns: void

void w\_Group::hide\_old()

Description: Sets the hide\_show\_ flag to hide for all the old components. returns: void

void w\_Group::receive()

Description: This function is called whenever this object is received from the application. It calls iupdate() on its interface\_element\_. returns: void

## FILES

w\_group.cc w\_group.hh

## 2.5.6.24 w\_Label

### NAME

w\_Label - Windui extensions to DUI\_Label.

### SYNOPSIS

```
#include "w_Label.H"
```

w\_Label.HH Contains definitions specific to w\_Label as modifications for duit sister class DUI\_label.

```
protected:
class WTStatic * interface_element_;
public:
WTStatic * interface_element() { return interface_element_; };
int icreate(w_View *aParent);
virtual int reposition(int X, int Y);
virtual int resize(int W, int H);
virtual int isize(int *W, int *H);
virtual void client_construct();
void no_ielement() { interface_element_ = 0;
hidden_or_shown_ = 0;};
void hide_component();
void show_component();
void make_window();
void really_show();
void receive();
```

### DESCRIPTION

These methods are extensions to the DUI\_Label(which see) class defined in the DUI Toolkit. They provide additional functionality required by the Windows DUI client (e.g. functions dealing directly with display of this object). It has a pointer to an associated object WTStatic(which see) which actually defines the Windows element.

### MEMBER FUNCTIONS

int w\_Label::icreate(w\_View \*aParent)

Description: This function is called to update an existing label if it has changed. It will also create a new label if this one has not been created yet. returns: void

void w\_Label::really\_show()

Description: This function checks hide\_show\_ and hidden\_or\_shown\_ to see if element must be explicitly displayed or hidden. This function is called on a final pass through a view's

instance hierarchy to display or hide the actual elements. The `hide_show_flag` is set when elements are created or updated. This is done so the view's components can change dynamically. The components always exist once created but may be removed or added to a view at any time during execution therefore components which are no longer part of a view are hidden instead of removed. Components can not be part of two different views at once. Although this is desirable. returns: void

`int w_Label::reposition(int X, int Y)`

Description: This function allows another object to ask this object to reposition itself given the passed coordinates. It adjusts the position of its title member and then of its windows element member. returns: int, 1 always

`int w_Label::resize(int W, int H)`

Description: This function allows for resizing of the object. returns: int, 1 always.

`int w_Label::isize(int *W, int *H)`

Description: This function returns its width and height into the arguments passed. returns: int, 1 always.

`w_Label::client_construct()`

Description: This is an addition to the constructor. It initializes the data members added in these extensions. returns: void

`void w_Label::receive()`

Description: This function is called whenever this object is received from the application. It calls `iupdate()` on its `interface_element_`. returns: void

## FILES

`w_label.cc w_label.hh`

### 2.5.6.25 w\_Selection

#### NAME

w\_Selection - Windui extensions to DUI\_Selection.

#### SYNOPSIS

```
#include "w_Selection.H"
```

w\_Selection.HH Contains definitions specific to w\_Selection as modifications for duit

```
protected:
int isCombo ;
class TListBox * interface_element_;
public:
    TListBox * interface_element() { return interface_element_; };
virtual int icreate(w_View *aParent);
virtual int reposition(int X, int Y);
virtual int resize(int W, int H);
virtual int isize(int *W, int *H);
virtual void client_construct();
virtual void no_ielement() { interface_element_ = 0;
hidden_or_shown_ = 0;};
virtual void hide_component() ;
virtual void show_component() ;
virtual void make_window();
virtual void really_show();
    virtual void receive();
```

#### DESCRIPTION

These methods are extensions to the DUI\_Selection(which see) class defined in the DUI Toolkit. They provide additional functionality required by the Windows DUI client (e.g. functions dealing directly with display of this object). It has a pointer to an associated object WTLListBox(which see) which actually defines the Windows element.

#### MEMBER FUNCTIONS

int w\_Selection::icreate(w\_View \*aParent)

Description: This function is called to update an existing object if it has changed. It will also create a new object if this one has not been created yet. returns: void

void w\_Selection::really\_show()

Description: This function checks hide\_show\_ and hidden\_or\_shown\_ to see if element must be explicitly displayed

or hidden. This function is called on a final pass through a view's instance hierarchy to display or hide the actual elements. The `hide_show_flag` is set when elements are created or updated. This is done so the view's components can change dynamically. The components always exist once created but may be removed or added to a view at any time during execution therefore components which are no longer part of a view are hidden instead of removed. Components can not be part of two different views at once. Although this is desirable. returns: void

`int w_Selection::reposition(int X, int Y)`

Description: This function allows another object to ask this object to reposition itself given the passed coordinates. It adjusts the position of its title member and then of its windows element member. returns: int, 1 always

`int w_Selection::resize(int W, int H)`

Description: This function allows for resizing of the object. returns: int, 1 always.

`int w_Selection::isize(int *W, int *H)`

Description: This function returns its width and height into the arguments passed. returns: int, 1 always.

`w_Selection::client_construct()`

Description: This is an addition to the constructor. It initializes the data members added in these extensions. returns: void

`void w_Selection::receive()`

Description: This function is called whenever this object is received from the application. It calls `iupdate()` on its `interface_element_`. returns: void

## FILES

`w_select.cc` `w_select.hh`

## 2.5.6.26 w\_Table

NAME

w\_Table - Windui extensions to DUI\_Table.

SYNOPSIS

```
#include "w_Table.H"
```

```
/* w_Table.HH * Contains definitions specific to w_Table as  
modifications for duit * sister class DUI_Table. * Kevin Convy 12/29/92  
*/
```

```
protected:
```

```
    friend class WTable;
```

```
class WTable * interface_element_;
```

```
int *column_widths;
```

```
int row_width_;
```

```
STRING *title_string_;
```

```
public:
```

```
    int column_width(int col);
```

```
    int row_width() { return row_width_; };
```

```
    WTable * interface_element() { return interface_element_; };
```

```
virtual int icreate(w_View *aParent);
```

```
virtual int reposition(int X, int Y);
```

```
virtual int resize(int W, int H);
```

```
virtual int isize(int *W, int *H);
```

```
char * column_title_string();
```

```
virtual void client_construct();
```

```
virtual void client_destruct();
```

```
virtual void no_ielement() { interface_element_ = 0;
```

```
hidden_or_shown_ = 0;};
```

```
virtual void hide_component() ;
```

```
virtual void show_component() ;
```

```
virtual void make_window();
```

```
virtual void really_show();
```

```
virtual void receive();
```

DESCRIPTION

These methods are extensions to the DUI\_Table(which see) class defined in the DUI Toolkit. They provide additional functionality required by the Windows DUI client (e.g. functions dealing directly with display of this object). It has a pointer to an associated object WTable(which see) which actually defines the Windows element.

MEMBER FUNCTIONS



int w\_Table::icreate(w\_View \*aParent)

Description: This function is called to update an existing object if it has changed. It will also create a new object if this one has not been created yet. returns: void

void w\_Table::really\_show()

Description: This function checks hide\_show\_ and hidden\_or\_shown\_ to see if element must be explicitly displayed or hidden. This function is called on a final pass through a view's instance hierarchy to display or hide the actual elements. The hide\_show\_ flag is set when elements are created or updated. This is done so the view's components can change dynamically. The components always exist once created but may be removed or added to a view at any time during execution therefore components which are no longer part of a view are hidden instead of removed. Components can not be part of two different views at once. Although this is desirable. returns: void

int w\_Table::reposition(int X, int Y)

Description: This function allows another object to ask this object to reposition itself given the passed coordinates. It adjusts the position of its title member and then of its windows element member. returns: int, 1 always

int w\_Table::resize(int W, int H)

Description: This function allows for resizing of the object. returns: int, 1 always.

int w\_Table::isize(int \*W, int \*H)

Description: This function returns its width and height into the arguments passed. returns: int, 1 always.

w\_Table::client\_construct()

Description: This is an addition to the constructor. It initializes the data members added in these extensions. returns: void

w\_Table::client\_destruct()

Description: Destructor additions for this class. returns: void

void w\_Table::receive()

Description: This function is called whenever this object is received from the application. It calls iupdate() on its interface\_element\_. returns: void

int w\_Table::column\_width(int col)

Description: Accesser function. returns: int, width of desired column.

char \*w\_Table::column\_title\_string()

Description: Creates a title string from all the names of the

columns plus padding(with spaces) for names that are shorter than their column's width. returns: char \*, title string.

## FILES

w\_table.cc w\_table.hh

### 2.5.6.27 w\_Text

NAME

w\_Text - Windui extensions to DUI\_Text.

SYNOPSIS

```
#include "w_Text.H"
```

w\_Text.HH Contains definitions specific to w\_Text as modifications for duit sister class DUI\_Text.

```
protected:
class WText * interface_element_;
public:
WText * interface_element() { return interface_element_; };
int icreate(w_View *aParent);
virtual int reposition(int X, int Y);
virtual int resize(int W, int H);
virtual int isize(int *W, int *H);
virtual void client_construct();
void no_ielement() { interface_element_ = 0;
hidden_or_shown_ = 0;};
void hide_component();
void show_component();
void make_window();
void really_show();
void receive();
```

DESCRIPTION

These methods are extensions to the DUI\_Text(which see) class defined in the DUI Toolkit. They provide additional functionality required by the Windows DUI client (e.g. functions dealing directly with display of this object). It has a pointer to an associated object WText(which see) which actually defines the Windows element.

MEMBER FUNCTIONS

void w\_Text::really\_show()

Description: This function checks hide\_show\_ and hidden\_or\_shown\_ to see if element must be explicitly displayed or hidden. This function is called on a final pass through a view's instance hierarchy to display or hide the actual elements. The hide\_show\_ flag is set when elements are created or updated. This is done so the view's components can change dynamically. The components always exist once created but may be removed or added to a view at any time during execution therefore components

which are no longer part of a view are hidden instead of removed. Components can not be part of two different views at once. Although this is desirable. returns: void

`int w_Text::reposition(int X, int Y)`

Description: This function allows another object to ask this object to reposition itself given the passed coordinates. It adjusts the position of its title member and then of its windows element member. returns: int, 1 always

`int w_Text::resize(int W, int H)`

Description: This function allows for resizing of the object. returns: int, 1 always.

`int w_Text::isize(int *W, int *H)`

Description: This function returns its width and height into the arguments passed. returns: int, 1 always.

`w_Text::client_construct()`

Description: This is an addition to the constructor. It initializes the data members added in these extensions. returns: void

`void w_Text::receive()`

Description: This function is called whenever this object is received from the application. It calls `iupdate()` on its `interface_element_`. returns: void

## FILES

`w_text.cc w_text.hh`

## 2.5.6.28 w\_Toggle

NAME

w\_Toggle - Windui extensions to DUI\_Toggle.

SYNOPSIS

```
#include "w_Toggle.H"

/* w_Toggle.HH Contains definitions specific to w_Toggle as
modifications for duit sister class DUI_toggle.

protected:
class TButton * interface_element_;
int isRadio_;
public:
TButton * interface_element() { return interface_element_; };
int icreate(w_View *aParent);
virtual int reposition(int X, int Y);
virtual int resize(int W, int H);
virtual int isize(int *W, int *H);
virtual void client_construct();
void no_ielement() { interface_element_ = 0;
hidden_or_shown_ = 0;};
void hide_component() ;
void show_component() ;
void make_window();
void really_show();
void iupdate();
void receive();
```

## DESCRIPTION

These methods are extensions to the DUI\_Toggle(which see) class defined in the DUI Toolkit. They provide additional functionality required by the Windows DUI client (e.g. functions dealing directly with display of this object). It has a pointer to an associated object WTCheckbox(which see) which actually defines the Windows element.

## MEMBER FUNCTIONS

int w\_Toggle::icreate(w\_View \*aParent)

Description: This function is called to update an existing object if it has changed. It will also create a new object if this one has not been created yet. returns: void

void w\_Toggle::iupdate()

Description: This function calls iupdate on its interface\_element\_.

returns: void

void w\_Toggle::really\_show()

Description: This function checks `hide_show_` and `hidden_or_shown_` to see if element must be explicitly displayed or hidden. This function is called on a final pass through a view's instance hierarchy to display or hide the actual elements. The `hide_show_` flag is set when elements are created or updated. This is done so the view's components can change dynamically. The components always exist once created but may be removed or added to a view at any time during execution therefore components which are no longer part of a view are hidden instead of removed. Components can not be part of two different views at once. Although this is desirable. returns: void

int w\_Toggle::reposition(int X, int Y)

Description: This function allows another object to ask this object to reposition itself given the passed coordinates. It adjusts the position of its title member and then of its windows element member. returns: int, 1 always

int w\_Toggle::resize(int W, int H)

Description: This function allows for resizing of the object. returns: int, 1 always.

int w\_Toggle::isize(int \*W, int \*H)

Description: This function returns its width and height into the arguments passed. returns: int, 1 always.

w\_Toggle::client\_construct()

Description: This is an addition to the constructor. It initializes the data members added in these extensions. returns: void

void w\_Toggle::receive()

Description: This function is called whenever this object is received from the application. It calls `iupdate()` on its `interface_element_`. returns: void

## FILES

w\_toggle.cc w\_toggle.hh

## 2.5.6.29 w\_View

### NAME

w\_View - Windui extensions to DUI\_View.

### SYNOPSIS

```
#include "w_View.H"
```

w\_View.HH Contains definitions specific to w\_View as modifications for duit sister class DUI\_View.

```
protected:
class WtWindow * interface_element_;
w_Component *old_component_;
w_Command *old_command_;
public:
WtWindow * interface_element() { return interface_element_; };
virtual int icreate();
void no_ielement() { interface_element_ = 0; };
virtual void client_construct();
void really_show();
```

### DESCRIPTION

These methods are extensions to the DUI\_View(which see) class defined in the DUI Toolkit. They provide additional functionality required by the Windows DUI client (e.g. functions dealing directly with display of this object). It has a pointer to an associated object WtWindow(which see) which actually defines the Windows element.

### MEMBER FUNCTIONS

int w\_View::icreate()

Description: This function either creates the window as necessary or updates it. In either case it calls the icreate funtion for its children. returns: int, 1 always.

void w\_View::really\_show()

Description: This function calls really\_show on the old and the new components to do the actual Windows showing of the interface\_elements\_ dependent on the hide\_show\_ flag. The hide\_show\_ flag is used because it might be the case that some widget has only changed its parent and would therefore be hidden for one parent and visible for another. This conflict is resolved by having the hide\_show\_ flag hold the desired attribute. returns: void

void w\_View::receive()

Description: Receive function for w\_View. This function is called when the view is received from the application it is also called by w\_Form::recieve(). returns:  
void

w\_View::client\_construct()

Description: This is an addition to the constructor. It initializes the data members added in these extensions. returns: void

## FILES

w\_view.cc w\_view.hh



### 2.5.6.30 w\_Widget

NAME

w\_Widget - Windui extensions to DUI\_Widget.

SYNOPSIS

```
#include "w_Widget.H"
```

```
protected:
```

```
int hidden_or_shown_;
```

```
char *logical_name_;
```

```
friend class WTWindow;
```

```
private:
```

```
static int dictionary_read_;
```

```
static class Dictionary *resource_dictionary_;
```

```
static struct stat current_file_stat_;
```

```
w_Widget * client_parent_;
```

```
private:
```

```
char *view_name();
```

```
public:
```

```
int read_resources();
```

```
int layout_;
```

```
int re_size_;
```

```
public:
```

```
char *absolute_resource_path();
```

```
char *relative_resource_path();
```

```
char *class_resource_path();
```

```
char *view_class_resource_path();
```

```
char *view_name_resource_path();
```

```
char *retrieve_named_resource(char *resource_name);
```

```
void client_parent(w_Widget *parent) { client_parent_ = parent; };
```

```
w_Widget * client_parent() { return client_parent_; };
```

```
void set_hidden_or_shown(int setting = 0) { hidden_or_shown_ = setting; };
```

```
virtual int reposition(int X, int Y) { return -1; };
```

```
virtual int resize(int W, int H) { return -1; };
```

```
virtual int isize(int *W, int *H) { return -1; };
```

```
virtual void setresized(int val) { if (client_parent_) { client_parent_->setresized(val);
```

```
}
```

```
re_size_ = val;
```

```
};
```

```
virtual int isresized() { return re_size_; };
```

```
virtual int icreate(class w_View *aParent) { return -1;
```

```
};
```

```
virtual int idestroy() { return -1; };
```

```
virtual int iupdate(w_View *aParent) { return -1; };
```

```
Windui(5) Last change: Wed Jan 5 17:53:10 1994 1
```

w\_Widget(5)

Gatec Manual

w\_Widget(5)

```

void layout(int direction) { layout_ = direction; };
int layout() { return layout_; };
int Rbackground(int *R, int *G, int *B);
int Rforeground(int *R, int *G, int *B);
char *Rname();
char *Ricon();
char *Rtitleposition();
char *Rfontname();
int Rfontheight();
int Rfontfixed();
int Rfontunderline();
int Rfontitalic();
int Rfontweight();
char *Rrepresentation();
char *Rlayout();
int Rdimensions(int *X, int *Y);
int Rexplicitdimensions(int **Xarray);
int Rcolumnwidths(int **Warray);
int Rhorizontalspacing();
int Rverticalspacing();
char *Ralignment();
int Rlength();
int Rwidth();
int Rheight();
int Ritemsshown();
char *Rdefaultvalue();
char *Rdoubleclick();
char *Rbitmap();
int Rwait();
int Rwaitedfor();
char *logical_name();
char *retrieve_value(char *resource_path);

```

## DESCRIPTION

These methods are extensions to the `DUI_Widget`(which see) class defined in the `DUI Toolkit`. They provide additional functionality required by the `Windows DUI client`. The primary purpose of these extensions is to provide access to `DUI resources` for all widgets. (see `DUI`).

## MEMBER FUNCTIONS

`w_Widget::read_resources()`

Description: This function reads the resource file and stores the paths and values as `Local_Atoms` in the `w_Widget's` resource dictionary. The resource file is expected to be named "`wres.res`" in the current directory. returns: `int`, 1 if success, -1 if failure.

`char *w_Widget::retrieve_value(char *resource_path)`

Description: This function retrieves the value for a given resource path. returns: char \*, value or NULL.

char \*w\_Widget::absolute\_resource\_path()

Description: This function builds an absolute path to this particular widget requesting an absolute path of it's parent which dominoes through the parent list. Absolute paths are of the form: <widget name | widget class name>[. <widget name | widget class name>]... An example path is: Make Award.first group.second group.Award number Paths are treated as caseinsensitive strings. This path overrides the relative path, and class path if any. returns: char \*, the path.

char \*w\_Widget::relative\_resource\_path()

Description: This function builds a relative path of the form: \*<widget name> Example: \*Award number This path overrides the class path if any. returns: char \*, the path.

char \*w\_Widget::class\_resource\_path()

Description: This function builds a class name path. Resource paths specifying entire classes are permitted. In this case they are applied to any widget belonging to that class, if their are no relative or absolute paths applying. returns: char \*,the path.

char \*w\_Widget::view\_name\_resource\_path()

Description: This function builds a view plus widget- name path: <view name>\*<widget name> returns: char \*, the path.

char \*w\_Widget::view\_class\_resource\_path()

Description: This function builds a view plus classname path: <view name>\*<classname> returns: char \*, the path.

char \*w\_Widget::view\_name()

Description: This function retrieves the view name for this widget. returns: char \*, the view name.

int w\_Widget::Rbackground(int \*R, int \*G, int \*B)

Description: Access function for "background" resource. returns: int 1 if there was a resource, 0 otherwise.

int w\_Widget::Rforeground(int \*R, int \*G, int \*B)

Description: Access function for "foreground" resource. returns: int 1 if there is a resource, 0 otherwise.

char \*w\_Widget::Rname()

Description: Access function for "name" resource. returns: char \*, the value or NULL.

char \*w\_Widget::Rdefaultvalue()

Description: Access function for "defaultvalue" resource. returns:

char \*, the value or NULL.

char \*w\_Widget::Rdoubleclick()

Description: Access function for "doubleclick" resource. returns: char \*, the value or NULL.

int w\_Widget::Rwait()

Description: Access function for "wait" resource. returns: true or false(1 or 0).

int w\_Widget::Rwaitedfor()

Description: Access function for "waitedfor" resource. returns: 1 if true, 0 if false.

char \*w\_Widget::Rbitmap()

Description: Access function for "bitmap" resource. returns: char \*, the value or NULL.

char \*w\_Widget::Ricon()

Description: Access function for "icon" resource. returns: char \*, the value or NULL.

char \*w\_Widget::Rtitleposition()

Description: Access function for "titleposition" resource. returns: char \*, the value or NULL.

char \*w\_Widget::Rfontname()

Description: Access function for "fontname" resource. returns: char \*, the value or NULL.

int w\_Widget::Rfontheight()

Description: Access function for "fontheight" resource. returns: int, the height, or 10.

int w\_Widget::Rfontweight()

Description: Access function for "fontweight" resource. returns: int, the value / 100 \* 100 % 1000.

int w\_Widget::Rfontfixed()

Description: Access function for "fontfixed" resource. returns: int 1 for true, 0 for false.

int w\_Widget::Rfontunderline()

Description: Access function for "fontunderline" resource. returns: int 1 for true, 0 for false.

int w\_Widget::Rfontitalic()

Description: Access function for "fontitalic" resource. returns: int 1 for true 0 for false.

char \*w\_Widget::Representation()

Description: Access function for "representation" resource.  
returns: char \*, the value or NULL.

char \*w\_Widget::Rlayout()

Description: Access function for "layout" resource. returns: char \*, the value or NULL.

int w\_Widget::Rdimensions(int \*X, int \*Y)

Description: Access function for "dimensions" resource. returns: int, 1 if value exists, 0 otherwise.

int w\_Widget::Rexplicitdimensions(int \*\*Xarray)

Description: Access function for "explicitdimensions" resource. returns: int 1 if there is a value, 0 otherwise.

int w\_Widget::Rcolumnwidths(int \*\*Warray)

Description: Access function for "columnwidths" resource. returns: int 1 if there is a value, 0 otherwise.

int w\_Widget::Rhorizontalspacing()

Description: Access function for "horizontalspacing" resource. returns: int, the value.

int w\_Widget::Rverticalspacing()

Description: Access function for "verticalspacing" resource. returns: int, the value.

char \*w\_Widget::Ralignment()

Description: Access function for "alignment" resource. returns: char \*, the value or NULL.

int w\_Widget::Rlength()

Description: Access function for "length" resource. returns: int, the value.

int w\_Widget::Rwidth()

Description: Access function for "width" resource. returns: int, the value.

int w\_Widget::Rheight()

Description: Access function for "height" resource. returns: int, the value.

int w\_Widget::Ritemsshown()

Description: Access function for "itemsshown" resource. returns: int, the value.

char \*w\_Widget::retrieve\_named\_resource(char \*resource\_name)

Description: This function performs the path search operations

needed by the resource accessor functions. returns: char \*, the value of the named resource or NULL.

char \*w\_Widget::logical\_name()

Description: function to return the name of the widget. The name can either be its supplied name or the resource replacement for supplied name. returns: char \*, the name.

## FILES

w\_widget.cc w\_widget.hh -

### 2.5.6.31 Session

NAME

Session - windui Session class.

SYNOPSIS

```
#include "Session.H"
```

```
class Session { protected:
    Session( char *programe );
    ~Session();
    int      status;
    int      running;
    istream*  inchannel;
    ostream*  outchannel;
    AppControl* thisapp;
    ConfigInfo* configuration;
    ofstream*  log_;
    static Session *instance();
    static Session *instance_;
public:
    static void send(Communication_Object* );
static void run();
static void poll();
    static int inerror();
    static int end();
    static ofstream& log();
    static void warning( const char *c );
    static void debug( const char *c );
    static void dodisconnect();
};
/* * Client_Session class definition. * */

class Client_Session: public Session { private:
    Client_Session(char *programe): Session(programe) {};
    ~Client_Session() {};
public:
    static int begin(char *appname, void (*efp)()=0);
};
/* * Server_Session class definition. * */

class Server_Session: public Session { private:
    Server_Session(char *programe): Session(programe) {};
    ~Server_Session() {};
public:
    static int begin(char *appname, void (*efp)()=0);
};
/* * Application_Session class definition. * */
```

```

class Application_Session: public Session { private:
    Application_Session(char *progrname): Session(progrname)
    {};
    ~Application_Session() {};
public:
    static int begin(char *appname, void (*efp)()=0);
}
DESCRIPTION

```

This class contains much of the code in the DUI Session(1) class, but has been modified to support serial communications under the MS Windows 3.1 environment.

## MEMBER FUNCTIONS

inline Session \*Session::instance()  
 Description: Accessor function for the one instance of Session.  
 returns: Session \*, the instance.

Session::Session(char \*appname)  
 Description: Constructor accepting the remote application name as argument. returns: void

void Session::send(Communication\_Object\* cobject)  
 Description: Function that sends a Communication object through the output channel. returns: void

Session::~~Session()  
 Description: Destructor for Session. Deletes channels, application name, and configuration info. returns: void

int Session::end()  
 Description: This member is a modified version of the Session(1) member. It runs the disconnect script "discon.scr" and terminates. returns: int -1 for error if it returns at all.

void Session::dodisconnect()  
 Description: This member does not appear in the Session(1) class. It just runs the "discon.scr" script and returns. returns: void

int Session::inerror()  
 Description: Status function. returns: int 1 if error, 0 otherwise.

void Session::run()  
 Description: This member is a modified version of the Session(1) member. It just calls Session::poll() because there is already an event loop in Windows and there would be a conflict if Session went into an endless loop waiting on



the application. returns: void

void Session::poll()

Description: This is a new member function. It checks the next character on port and if it is a "(", it sets blocking mode and reads in the object, otherwise it returns. It also sends a neutral AppControl object every 10000 times it is called for channels that need activity in order to stay live. returns: void

ofstream& Session::log( )

Description: Accessor function. returns: ofstream&, a log file stream.

void Session::warning( const char \*c )

Description: Writes message to log. returns: void

void Session::debug( const char \*c )

Description: Writes message to log. returns: void

int Client\_Session::begin(char \*appname, void (\*efp)

Description: This member has been modified to support serial communications. returns: void.

## FILES

session.c session.h

### 2.5.6.32 SerialBuf

#### NAME

SerialBuf - streambuf derivative for a Windows serial port.

#### SYNOPSIS

```
#include "SerialBuf.H"
```

```
class SerialBuf: public ChannelBuf { public:
SerialBuf();
virtual ~SerialBuf();
virtual int connect(ConfigInfo *config);
virtual int reconfigure(ConfigInfo *config);
virtual int reconfigure(const DCB*);
virtual COMSTAT *getlasterror();
int blocking();
int blocking(int);
protected:
private:
char *port();
int fd();
int opened();
SerialBuf *verbose(int );
virtual int disconnect();
virtual int overflow(int c = EOF);
virtual int underflow();
virtual int sync();
virtual int doallocate();
void error(const char *);
void sys_error(const int);
char *_port;
int _fd;
int _opened;
int _close;
int _blocking;
int _verbose;
COMSTAT error_status;
}
```

#### DESCRIPTION

This class implements a streambuf for a serial comm port under the windows operating system.

#### MEMBER FUNCTIONS

int SerialBuf::connect(ConfigInfo \*config)

Description: This function retrieves the serial configuration information from the passed ConfigInfo object and opens the

comm port. returns: int 1 if success, -1 if error.

int SerialBuf::reconfigure(ConfigInfo \*config)

Description: This function reconfigures the serial line based upon the passed ConfigInfo object. returns: int 0 if successful, -1 if failure.

int SerialBuf::reconfigure(const DCB \*newdcb)

Description: This allows reconfiguring using a DCB structure. returns: void

COMSTAT \*SerialBuf::getlasterror()

Description: This function clears the last communications error state and reports on the following error or status states: CE\_OVERRUN, CE\_TXFULL, CSTF\_XOFFHOLD, CSTF\_XOFFSENT returns: COMSTAT \*, the error status returned.

int SerialBuf::disconnect()

Description: Flushes the port and closes it. returns: void

SerialBuf \*SerialBuf::verbose(int verbose )

Description: Sets verbose error reporting. returns: void

SerialBuf::SerialBuf()

Description: Empty constructor. returns: void

int SerialBuf::fd()

Description: Accessor function. returns: int, the file descriptor.

int SerialBuf::blocking()

Description: Accessor function. returns: int, the blocking state.

int SerialBuf::blocking(int ablocking)

Description: Sets the blocking state. returns: int, the new blocking state.

int SerialBuf::opened()

Description: Accessor function. returns: int, the open status.

char \*SerialBuf::port()

Description: Accessor function. returns: char \*, the name of the port.

void SerialBuf::error(const char \*msg)

Description: output error message "msg". returns: void

void SerialBuf::sys\_error(const int retcode)

Description: Output string description of system error with code "retcode". returns: void

SerialBuf::~~SerialBuf()

Description: Destructor, calls disconnect and deallocates memory. returns: void

int SerialBuf::doallocate()

Description: Allocates io buffers. returns: void

int SerialBuf::overflow(int c )

Description: Write put buffer to serial port. returns: int, number of chars written.

int SerialBuf::sync()

Description: Calls underflow() and overflow(). returns: int, return value of overflow() (number of chars written).

int SerialBuf::underflow()

Description: Reads from serial port. If blocking is set calls Communications\_Script on script file "pause.scr" which it expects to find in the current directory 5 times attempting a read between each retry if still nothing on the port after 5 times it returns EOF, otherwise if there is something on the port reads as much as it can and returns next character in the get buffer. returns: int, next character in the get buffer.

## FILES

serialbu.c serialbu.h

---

---

## SECTION 3 The GATEC Database Software

---

The software that comprises the establishment and access to the GATEC 2 database is located at \$CVSROOT/narqdb in the GATEC development environment.

The second and third sections of this database software description deal with NORA and NARQ libraries, respectively. Those sections concentrate on the content and intended use of each of the given libraries. The fourth section describes the files in the development environment and instructions for constructing the libraries. The fifth section contains a few sample applications showing how NARQ and NORA can be used in a C++ application.

---

### 3.0.1 NARQ & NORA

---

NARQ is an acronym for the most commonly used objects in the GATEC procurement process; Notes, Acquisitions, Requests for quote, and Quotes. Specifically, it is one of two libraries that is available to C++ programmers that allows access to database records. The NARQ library consists entirely of compiled C++ object code originally generated from descriptions of the GATEC database objects.

NORA is an abbreviation of NARQ Oracle. It is the second of two libraries used to access an Oracle database. However, unlike the NARQ library, the NORA library contains no GATEC-specific information. It represents a logical separation of the application-specific objects from the Oracle-specific objects. It primarily contains object code providing functionality analogous to SQL statements.

The combination of the NARQ and NORA libraries provides the ability to query and update a GATEC database without the need for writing a single line of SQL code.

---

## 3.1 NORA Principles

---

The NORA library was designed to provide an object interface to an Oracle database without the requirement of having to know the particulars of the schema representation nor the rigors of Embedded-SQL programming. The library provides some of the capabilities of the SQL language. All language features are not implemented due to time considerations and usage needs. However, an SQL gateway class is provided to allow the use of SQL statements when the existing classes are insufficient.

The ability to query, join and update is represented. The ability to remove records is apparently included, but the NORA interface only provides the ability of tagging information as deleted and, for data integrity reasons, has intentionally omitted delete capability. Related to this fact is the library's management of updates. In similar fashion to delete operations, update operations do not overwrite information. Instead, current information is tagged as obsolete in favor of updated information. Despite the space penalty resulting from this design decision, the benefit of information tracking and accountability is especially useful during the debugging process.

---

### 3.1.1 NORA Classes

---

The classes that comprise the NORA library have a very strong correlation to a number of keywords found in the SQL language. Similarly, NORA objects are intended to be used together to construct valid database operations in the same manner that a syntactically correct SQL statement would be formed. The remainder of this section identifies the classes that are found in the NORA library and explains their relevance and relation to other classes. For details on the programming interfaces and private data members, please refer to the NORA man pages, the *NORA Design Reference* manual or the appropriate header files.

#### *Connection*

The Connection class is responsible for managing the connection with the host (Oracle) database. Generally, only one instance is required in an application, but it is possible to open several connections to several remote and local databases or as several different user names. In order to make a successful database connection, a valid Oracle user name and password combination pair is required along with a valid remote host string if accessing a

remote database.

The only other significant capabilities associated with the Connection class are the commit and rollback functions that are identical in use to the SQL statements of the same name. Closing a connection by freeing the object or calling the disconnect member function will also initiate an implied commit of any outstanding transactions associated with the Connection.

### *Database*

The Database class is the origin from which the Connection class originated and is intended to provide backwards compatibility with early GATEC applications that relied on its presence. Where the Connection class makes it possible for manage several, simultaneous database connections, the original Database class had the provision for only a single active database connection. The Database class now exists primarily as a front end to the an underlying Connection object. Its functions are identical to those of a Connection object and simply call the associated function. The Database class is a static class and there should only be one instance of it in any application.

The Database class does serve a useful purpose, however. Many NORA objects take an optional Connection object which indicates which database connection is to be used. In the case where only a single database connection is made, or is dominant over several other database connections, the Database class identifies the *default* Connection that should be used if not supplied in the various object constructors.

### *DBObject*

The DBObject class is an abstract base class that is responsible for managing the interface to the host (Oracle) database library routines. A majority of the Oracle-specific code can be found in the member functions that populate this class. The only public member functions associated with it are related to debugging, monitoring error messages generated and examining the SQL statements generated by any derived objects. The various protected member functions contain code that interface with the Oracle Call Interface (OCI) library thus enabling derived classes to manipulate data contained in the Oracle database. Cursor management, query parsing and execution are among the functions available in this class.

### *Query*

The Query class is derived from the DBObject class and is itself an abstract base class. It is the base from which the three (current)

types of query structures are derived. It is uniquely responsible for managing such low-level activity as binding memory locations and cursor management.

### *SimpleQuery*

A SimpleQuery object was the first Query object developed and provides basic single table or single join query capability. In order to construct a query it requires at least one Table object and one Condition object (described below). It is usually unnecessary to call any of the member functions provided in this class since many are called by either the base class or the FetchedRows class (also see below).

### *ComplexQuery*

The ComplexQuery class is the workhorse query class and can manage any number of Table, Join (see below) and Condition (see below) objects. Unlike the SimpleQuery class, the ComplexQuery class is modifiable in the sense that new queries can be generated by adding or removing objects. Member functions are included that all the addition and removal of Table, Join or Condition objects. However, the class does not provide robust syntax checking of its objects. This task is the responsibility of the programmer. If a Table object were to be removed without removing a related Join object, an error message is sure to be generated during the SQL statement parsing.

Other than the ability of managing any number of query-related objects, ComplexQuery is similar to the SimpleQuery class in its association to the FetchedRows class.

### *ImmediateQuery*

This class will accept a valid SQL query (SELECT statement) that is free of wildcards in the returned columns list. The returned columns are stored in a QueryResult (see below) object that, along with the SQL query string, is a required parameter in the constructor. Like the above two derived Query classes, the ImmediateQuery class is intended for use with the FetchedRows class.

### *Table*

Like the Query class, the Table class is an abstract base class derived from the DBObj class. The necessity of the Table class is due to the OCI routines inability to bind to tables. Consequently, the Table class is used to associate table relationships and table-column relationships. Specifically, a Table object and its associated Column (see below) is similar to an Oracle table



description; an Oracle table is equated to a Table object.

Table objects are the basis for all of the objects defined in the NARQ library. Every class in the NARQ library is derived from the Table class and is representative of an Oracle database table.

The member functions in the Table class deal primarily with the management of the contained Column classes. However, this is usually managed by the derived classes. Beyond the management functions, the useful functions provide the ability to insert a new record or update/remove a fetched record, the ability to “dump” the contents of the table (current row/record) to a string, including to a CDF format and the ability to lock a table for exclusive use. Locking a table is useful for long, complex operations that rely on ensuring that no other database users can make changes to the table until the operation completes.

### *Dual*

The Dual class is the first of only two specialized NORA class objects that are derived from the Table class; QueryResult is the second. It is the function equivalent to the Oracle Dual table. It's only intended purpose was in conjunction with the DateColumn class to derive dates in a multitude of formats, including Julian.

### *QueryResult*

Because the Query class requires a Table object to bind database information to memory locations, the QueryResult class was designed to act as the repository for the information. However, because it is not actually representative of an Oracle table, several member functions, such as table locking or committing changes, have had their functionality removed.

### *Join*

The Join class is used by the SimpleQuery and ComplexQuery classes to define the table relationships to be used in the associated query. From an SQL vantage point, the join relationship is defined in the WHERE clause of an SQL statement. In addition to references to two Column objects defined in two separate (or a single) Table object(s), a Boolean comparison operation must be specified to define the relationship between the two Tables. However, it should be noted that, using the ComplexQuery class, multiple Join relationships can be defined between two or more Table objects.

### *Condition*

In addition to the Join class, the Condition class is logically used

to define the query condition in the WHERE clause of an SQL statement. Its basic purpose is to manage any number of Expression objects (the actual qualifications) as well as providing the interface to specifying the return order of the query rows. A Condition object is required by each Query class. However, a Condition object is not required to specify any Expressions. In this case, the query will return all rows associated with the Table or Table joins.

### *Expression*

The Expression class is the remaining NORA objects class (in addition to the Join and Condition classes) that is used to construct the WHERE clause portion of the SQL statement constructed by a Query class. A number of constructor classes provide column-to-column comparisons as well as column-to-string and column-to-number comparisons along with “is null” and “exists” comparisons.

### *Column*

The Column class is an abstract base class from which several classes related to the various Oracle data types are derived. Because Column objects are generally owned by a Table object, member functions are provided to reference back to the Table object as well as to provide the Table object with information useful for binding the private data area during database operations. The only significant remaining member functions allow for extracting values from and assigning values to the Column object.

It is unlikely that user applications will have any need to directly instantiate any derived Column object. The NARQ library contains generated code that is responsible for managing the Table/Column relationships. Each NARQ object class is responsible for instantiating the Column objects associated with each Table object.

### *CharColumn*

Derived from the Column class, the CharColumn class is analogous the Oracle CHAR data type. The only additional functionality provided by the class is the ability to assign string values to the Column object.

### *DateColumn*

Derived from the Column class, the DateColumn class is representative of the Oracle DATE column type. Like the CharColumn class, additional functionality is provided to allow the assignment of date values in any format known to Oracle. In addition, access functions controlling the output date format as

well as providing access to each segment of the date (i.e. hour, minute, day, month, year, century, etc.)

#### *FloatColumn, LongColumn, NumberColumn*

The FloatColumn, LongColumn and NumberColumn classes are essentially identical with the only difference in the precision of the number information held by each. Each of the three classes is based on the Oracle NUMBER data type. Value assignment is accomplished by numeric or string values.

#### *RawColumn*

The Oracle RAW data type allows for any type of information (including binary) to be stored in the field. The RawColumn class is nearly identical to the CharColumn class though it is bound to a different type of underlying data store. The present GATEC implementation (schema) does not have any instance of raw data. Consequently, programmatic support of this class is not yet fully developed. It has been allocated and addressed for the sake of completeness.

#### *RowID*

Oracle allows a query to return a row identifier that can be used to reference the record directly. However, the numbers generated by Oracle cannot be guaranteed to be consistent across sessions. To date, the GATEC development has not yet relied on its implementation and, as such, it has not yet fully undergone a full development cycle. Caution is recommended regarding its use.

#### *FetchedReaders*

The FetchedReaders class is used with Query objects to manage the iteration through the list of returned rows resulting from the query. Instantiation of a FetchedReaders object will initiate (execute) the associated query. Additional member functions allow fetching the next row, restarting the query and generating a count of the number of rows that the query will return.

It should be noted that is the FetchedReaders class does not allow reverse iteration (stepping backwards) through the set of returned rows. As a future implementation note, this capability should be possible through the use of the RowID class.

#### *FetchedReaderGroup*

Like the FetchedReaders class, the FetchedReaderGroup class manages the returned rows of a Query object. However, unlike the iterative nature of the FetchedReaders class, the FetchedReaderGroup class (though

not yet implemented) is intended to manage sets of objects representing the entire number of returned rows or a specified maximum number of returned rows. The object class (again, not yet implemented) should offer some performance increase in contrast to the FetchedGroup object, but it will be difficult to overcome the burdensome memory requirement that seems unavoidable.

### *Sequence*

The Sequence class is used to extract values from the sequences defined within Oracle. A sequence is a resource that returns a series of incremental (or decremental) values guaranteeing unique, successive values until the sequence reaches its maximum (minimum) value or specified limit. At such point, the sequence will reinitialize itself back to its initial value or return error values for each request depending on how the original sequence was constructed.

### *ivList*

The ivList class is similar to a template list class implementation and is intended for use by the NORA library objects. It was obtained from the library implementation provided as part of the Stanford InterViews interface project.

---

## 3.1.2 Limitations

---

The major limitation of the NORA library is its incomplete implementation of the SQL language. There are a number of useful SQL function calls that have no counterpart in the NORA library. Their use must be incorporated through new, custom code or passed through the Immediate Query class. Additionally, the current incarnation provides no support for Data Definition Language (DDL) statements such as CREATE, DROP, GRANT and REVOKE. Though it would be a fairly simple task to support any SQL statement by modification of ImmediateQuery class (which limits the use of only SELECT statements), the lack of structured class support is a negative point. While DDL statements are of limited use in the GATEC application, their absence is a major detraction for the use of NORA in future projects. A third (considerable) limitation is the library's lack of support for Oracle's procedural language, PL/SQL.

---

## 3.1.3 Detailed NORA Class Descriptions

---

### 3.1.3.1 CharColumn

The NORA classes are described in the following pages.

#### NAME

CharColumn - wrapper class for Oracle CHAR column datatype

#### SYNOPSIS

```
#include <nora/CharColumn.h>
```

#### DESCRIPTION

The CharColumn class provides an interface to Oracle columns defined as type "char." At run-time it is usually bound to specific named column and assumes information and data related to the column.

#### CONSTRUCTORS

In almost all cases, objects of this type are created as a result of creating another object; i.e. Table or QueryResult. However, use of the Dual class provides the opportunity to directly instantiate this type of object.

```
CharColumn(Table* t, unsigned size, char* name, char* value)
```

Instantiation of a CharColumn object requires an associated table. The size parameter determines the maximum size of the contents of the column. It is normally consistent with the Oracle schema definition and is currently limited to no more than 255 characters. The name parameter should match the name of the database column and the (optional) value parameter initializes the object's contents.

```
CharColumn(Table* t, CharColumn* cc)
```

Copy constructor

#### MEMBER FUNCTIONS

```
const char* contents(boolean)
```

This function returns a string containing the name and value of the object. The optional parameter determines whether the output follows the CDF standard and defaults to true if not given.

```
int oratype()
```

Returns information about the type of data contained in the object.

This is useful when working with Column objects.

`const char* value()`

Returns a string containing the current value of this object.

`const char* db_value()`

Used by other NORA objects for use with Oracle-specific operations.

`void* address()`

Used by other NORA objects for use with Oracle-specific operations.

`void operator= (const char*)`

Assignment operation used to set the value of the object.

`void assign_value(const char*)`

Alternative means of assignment. Used to set the value of the object.

`void assign_char_value(const char*)`

Alternative means of assignment. Used to set the value of the object.

`void empty()`

This function clears the contents of the object and restores its internal state.

`unsigned length()`

Returns information about the length of the object's contents.

## SEE ALSO

Column (3N), Dual(3N), Table(3N), QueryResult(3N)

### 3.1.3.2 Column

#### NAME

Column - Abstract base class for Oracle column types

#### SYNOPSIS

```
#include <nora/Column.h>
```

#### DESCRIPTION

The Column class is a base class from which specific typed columns are derived from.

#### CONSTRUCTORS

The constructor for the Column class is protected, thus preventing direct instantiation. The presence of several "pure" virtual functions also impose requirements on derived classes. Derived classes must take care to define these functions.

#### MEMBER FUNCTIONS

const char\* name()

Returns string containing the name of the Oracle table column to which this object will be bound at run-time.

Table\* table()

Returns pointer to Table object "owning" this Column object.

int oratype()

This is pure virtual function that returns information about the type of data contained in the object. The value returned is Oracle-defined.

const char\* db\_value()

This is a pure virtual function that returns the contents of this object for use by Oracle-specific (OCI) library routines.

void\* address()

This is a pure virtual function that returns a pointer the object's contents intended for use by Oracle-specific (OCI) library routines.

unsigned length()

This is a pure virtual function that returns the length of the object's contents intended for use by Oracle-specific (OCI) library routines.

`const char* contents(boolean)`

This is a pure virtual function that returns a string containing the name and value of the object. The parameter determines whether the output follows the CDF standard. `const char* value()`

This is a pure virtual function that returns a string containing the current value of the object.

`void assign_char_value(const char*)`

This is a pure virtual function that accepts the contents of a character string as the value assigned to this object.

`void assign_null()`

In Oracle terms, a NULL column has no value. This function will assign a NULL value to the object's contents.

`void ignore(boolean)`

Setting the parameter to true prevents this Column from being fetched when used in a Query.

`boolean ignore()`

Returns information that specifies whether the column is fetched if used in a Query.

`boolean null()`

Returns information that indicates whether the object's value is set to NULL.

`boolean modified()`

Returns information indicating whether the contents have been altered by a user process. Updates made by the Query classes are not applicable.

`unsigned size()`

Returns information about the size of the object's contents.

## SEE ALSO

Table (3N), Query (3N), SimpleQuery (3N)



### 3.1.3.3 DBObjct

#### NAME

DBObject - manages interface to host (Oracle) database library routines

#### SYNOPSIS

```
#include <nora/DBObject.h>
```

#### DESCRIPTION

The DBObjct class is an abstract base class that contain routines that interface with the Oracle Call Interface (OCI) routines enabling derived class to manipulate Oracle databases via the Oracle library calls defined in this class.

#### CONSTRUCTORS

Because this is an abstract base class, the constructor is protected, thus preventing direct instantiation of objects of this type.

#### MEMBER FUNCTIONS

boolean debug()

Returns information about the current state of debugging operations.

void debug(boolean)

Define whether debugging code should be turned on/off.

const char\* ora\_error\_msg()

If an Oracle error has occurred, this function will return the Oracle-generated error message corresponding to the fault.

const char\* error\_msg()

If a user-specified error message has been provided, this function will return a string containing the error message.

char\* show\_sql()

Returns a string containing the SQL code that was generated by a derived object.

#### PROTECTED MEMBER FUNCTIONS

These functions are available only to derived classes.

`cursor* cursor()`

This function returns the pointer to an Oracle-defined structure that contains information needed by the OCI routines.

`void parse(char*)`

This function is used by derived Query classes and is a required first step in the process of extracting data from the database. The parameter should be a valid SQL query string.

`void parse()`

This function is used by derived Query classes and is a required first step in the process of extracting data from the database. The query to be parsed is the string currently defined in the object.

`void execute()`

This function is used by derived Query classes to "start" a query after it has been parsed. It does not fetch any values from the database.

`boolean commit()`

This function is used by derived Table classes to commit data to the database. Note that the transaction is not permanent until a commit is invoked in the Database class (unless the autocommit mode is turned on.)

`void abort()`

This function is used by derived Query classes to cancel a query in progress.

`void restart()`

This function is used by derived Query classes. It provides the ability to cancel a query and restart it.

`void fetch()`

This function is used by the derived Query classes. After a query has been parsed and executed, this call fetches the returned row(s) from the database.

`int ora_error_val()`

Returns the error value returned by OCI calls. This should be used in conjunction with the `ora_error()` function call to determine when errors have occurred.

`boolean ora_error()`

Returns information about whether an Oracle error has occurred.

`void error_msg(char*)`

This function allows a derived object to embed application-

specific error messages.

`void clear_string()`

Clears the buffer containing generated SQL code.

#### SEE ALSO

Connection (3N), Database (3N), Table (3N), Query (3N)

### 3.1.3.4 Connection

#### NAME

Connection - manages connections to host (Oracle) databases

#### SYNOPSIS

```
#include <nora/defs.h> #include <nora/Connection.h>
```

#### DESCRIPTION

The Connection class is the singular method used to establish connections with an Oracle database. This is also the mechanisms that allows for multiple Oracle logons in the same user process. Once an Oracle connection is established, this class is responsible for committing or rolling back outstanding transactions on a per Connection basis.

#### CONSTRUCTORS

Connection(char\* username, char\* passwd, char\* remotedb)

The constructor's parameters allow the user to specify the username and password pair for a valid ORACLE user. The remotedb parameter is optional and defaults to the local machine or any remote database defined in the user's environment. (See Oracle manual for specifics). The connected() member function will indicate whether a successful connection was established.

#### MEMBER FUNCTIONS

boolean connected()

Returns information about whether an active database connection exists.

void disconnect()

Close the existing database connection. All uncommitted transactions are committed. To disconnect without committing outstanding transactions, use the rollback() member function before disconnecting.

const char\* username()

Returns a string indicating the Oracle user name used to connect to the database.

boolean commit()

Commit all outstanding transactions. Returns information about the success of the operation. If false is returned, the ora\_error\_msg() function indicates the reason for the failure. If the connection is configured to automatically commit transactions (see autocommit() below), this function will have no effect.

boolean rollback()

Rollback all outstanding transactions. Returns information about the success of the operation. If false is returned, the ora\_error\_msg() function indicates the reason for the failure. If the connection is configured to automatically commit transactions (see below), this function will have no effect.

boolean autocommit()

Returns information about whether the connection automatically commits individual transactions at the Table level (see Table(3N)). This is typically set to false.

void autocommit(boolean value)

Defines whether the connection will automatically commit individual transactions.

const char\* ora\_error\_msg()

If an Oracle error has occurred, this function will return the Oracle-generated error message corresponding to the fault.

cursor\* lda()

This function provides access to the Oracle-defined logon data area used by the Oracle Call Interface (OCI) library calls. This is useful for constructing specialized database access routines that use OCI calls.

boolean debug()

Returns information about the current state of debugging operations.

void debug(boolean value)

Define whether debugging code should be turned on/off.

### 3.1.3.5 Database

#### NAME

Database - establishes default connection to host (Oracle) databases

#### SYNOPSIS

```
#include <nora/defs.h> #include <nora/Database.h>
```

#### DESCRIPTION

The Database class was originally the only means to connect to an Oracle database. However, it only allowed one active connection. It has been superceded by the Connection class. The Database class now exists as a front end to a single Connection and serves a useful purpose in specifying the primary (or default) user connection.

#### CONSTRUCTORS

This is a static class with no public constructor. To open a Connection to a database, an instance must be first be created. Once created, the connect member function is used to open the connection. An example follows:

```
Connection* oracle_user = new Connection("scott", "tiger");
Database* db = Database::instance();
if (db->connect(oracle_user)) { cout << "Connection established"
<< endl;
}
```

#### MEMBER FUNCTIONS

With the exception of the connect() and connected() member functions, these functions assume that a Connection object (open Oracle connection) exists.

boolean connect(Connection\*)

This function is called to define the default Connection.

boolean connected()

Returns information about whether the Connection is active.

void disconnect()

Close the existing database connection. All uncommitted transactions are committed.

`const char* username()`

Returns a string indicating the Oracle user name associated with the Connection.

`boolean commit()`

Commit all outstanding transactions. Returns information about the success of the operation. If false is returned, the `ora_error_msg()` function indicates the reason for the failure. If the connection is configured to automatically commit transactions (see below), this function will have no effect.

`boolean rollback()`

Rollback all outstanding transactions. Returns information about the success of the operation. If false is returned, the `ora_error_msg()` function indicates the reason for the failure. If the connection is configured to automatically commit transactions (see below), this function will have no effect.

`boolean autocommit()`

Returns information about whether the connection automatically commits individual transactions at the Table level (see Table(3N)).

`void autocommit(boolean value)`

Defines whether the connection will automatically commit individual transactions.

`const char* ora_error_msg()`

If an Oracle error has occurred, this function will return the Oracle-generated error message corresponding to the fault.

`cursor* lda()`

This function provides access to the Oracle-defined logon data area used by the Oracle Call Interface (OCI) library calls. This is useful for custom database access routines.

`boolean debug()`

Returns information about the current state of debugging operations.

`void debug(boolean value)`

Define whether debugging code should be turned on/off.

`static Database* instance()`

This function returns a pointer to the static Database instance.

SEE ALSO

### 3.1.3.6 Condition

Connection (3N), Table (3N)

NAME

Condition - Equivalent to Oracle WHERE clause

SYNOPSIS

```
#include <nora/Condition.h>
```

DESCRIPTION

The Condition class is the framework for defining the WHERE clause of a generated query. It is required by both the SimpleQuery and ComplexQuery classes.

CONSTRUCTORS

Condition()

An empty constructor can be equated to an empty where clause. In this case, all rows are returned.

Condition(Expression\*)

The Expression parameter is used as the first clause in the WHERE clause.

MEMBER FUNCTIONS

boolean defined()

Returns information about whether any Expressions have been defined. Basically, this indicates whether the SQL WHERE clause is defined.

const char\* expr\_stmt()

Returns string containing only the expressions that would be included in the generated SQL clause.

const char\* order\_stmt()

Returns string containing only the ORDER BY portion of the generated SQL clause.

const char\* statement()

Returns string containing the complete SQL for the constructed WHERE clause.



`void reset_cond()`  
Remove all defined Expression relationships.

`void reset_order()`  
Remove any defined row ordering.

`void reset()`  
Purge all definitions; remove all contained Expressions and row order relationships.

`void and(Expression*)`  
Add an Expression to the Condition. If the Expression is the first Expression, no special functions are performed. If Expressions already exist, the new Expression is prefixed with the AND keyword.

`void or(Expression*)`  
Add an Expression to the Condition. If the Expression is the first Expression, no special functions are performed. If Expressions already exist, the new Expression is prefixed with the OR keyword.

`void and(Condition*)`  
Add in the Expressions contained in the Condition parameter. If no Expressions exist, the contents are copied exactly. If Expressions already exist, the AND keyword is prefixed.

`void or(Condition*)`  
Add in the Expressions contained in the Condition parameter. If no Expressions exist, the contents are copied exactly. If Expressions already exist, the OR keyword is prefixed.

`void order_by(Column*)`  
Define the ordering of the returned rows of the Query that will use this Condition. Subsequent invocations define the secondary columns for the ordering.

`void and(Expression* e1, LogicalOp l, Expression* e2, ...)`  
Short cut to using the `and()` and `or()` functions. AND is prefixed to beginning of clause. Accepts variable number of parameters. Subsequent parameters should alternate between LogicalOp and Expression. The list must be terminated by a NOOP LogicalOp.

`void or(Expression* e1, LogicalOp l, Expression* e2...)`  
Short cut to using the `and()` and `or()` functions. OR is prefixed to beginning of clause. Accepts variable number of parameters. Subsequent parameters should alternate between LogicalOp and Expression. The list must be terminated by a NOOP LogicalOp.

void and(Condition\* c1, LogicalOp l, Condition\* c2, ...)

Short cut to using the and() and or() functions. AND is prefixed to beginning of clause. Accepts variable number of parameters. Subsequent parameters should alternate between LogicalOp and Condition. The list must be terminated by a NOOP LogicalOp.

void or(Condition\* c1, LogicalOp l, Condition\* c2...)

Short cut to using the and() and or() functions. OR is prefixed to beginning of clause. Accepts variable number of parameters. Subsequent parameters should alternate between LogicalOp and Condition. The list must be terminated by a NOOP LogicalOp.

void order\_by(Column\* c1, Column\* c2, ...)

Short cut to specifying row ordering. Variable number of parameters are accepted, all of type Column. The list must be terminated by a null Column reference.

## PROTECTED MEMBER FUNCTIONS

void build\_cond(char\* and\_or, Condition\* c1, LogicalOp l, Condition\* c2, va\_list ap)

Construct SQL clause using known Conditions

void attach\_va\_cond(va\_list va\_l)

Maintain internal Condition list

void build\_expr(char\* and\_or, Expression\* c1, LogicalOp l, Expression\* c2, va\_list ap)

Construct SQL clause using known Expressions

void attach\_va\_expr(va\_list va\_l)

Maintain internal Expression list

## SEE ALSO

Expression (3N)

### 3.1.3.7 Dual

#### NAME

Dual - Equivalent to Oracle "Dual" table

#### SYNOPSIS

```
#include <nora/Dual.h>
```

#### DESCRIPTION

The Dual class is derived from the Table class and was designed for use with the DataColumn class.

#### CONSTRUCTORS

Dual()  
The constructor takes no arguments.

#### MEMBER FUNCTIONS

Because the Dual class is derive from the abstract Table class, several virtual functions are defined, but have no functionality (contain no code.) Modifications to the "dual" table are not allowed.

const char\* name()  
Returns a string containing the the value, "Dual."

#### SEE ALSO

Column (3N), DataColumn (3N), Table (3N)

### 3.1.3.8 ComplexQuery

#### NAME

ComplexQuery - Modifiable Query class

#### SYNOPSIS

```
#include <nora/ComplexQuery.h>
```

#### DESCRIPTION

The ComplexQuery class is derived from the Query class and allows modification to the object's contents to effectively reconstruct the underlying SQL query. There is NO error checking for badly formed query constructs.

#### CONSTRUCTORS

ComplexQuery(Table\*, Condition\*, boolean uniq, ReturnResult results)

Single table query. This constructor is the simplest way to begin constructing a query.

ComplexQuery(Connection\*, Table\*, Condition\*, boolean unique, ReturnResult results)

Identical to the above single table query. However, instead of using the connection defined by the static Database instance, a separate Connection can be used instead.

ComplexQuery(Table\* t1, Condition\* c, Table\* t2, Join\* j, boolean uniq, ReturnResult results)

This constructor is an identical interface to Simple-Query allowing for a simple migration for queries that need to expand in scope.

ComplexQuery(Connection\*, Table\* t1, Condition\* c, Table\* t2, Join\* j, boolean uniq, ReturnResult results)

Identical to the above SimpleQuery interface with the provision for specifying a Connection other than the default set in the Database instance.

ComplexQuery(TableList\* t, ConditionList\* c, JoinList\* j, boolean uniq, ReturnResult results)

This is the fastest way to construct a query since the generation of the lists of Tables, Conditions, and Joins is already in place.

## MEMBER FUNCTIONS

unsigned count()

Returns the number of rows that would be fetched by the query.

void add(Table\*)

Insert a Table reference into the internal Table list.

void add(Condition\*)

Insert a Condition reference into the internal Condition list.

void add(Join\*)

Insert a Join reference into the internal Join list.

void add(Table\* t, Condition\* c, Join\* j)

Short-cut method to insert references to a Table, Condition, and Join objects into their respective internal lists.

boolean remove(Table\*)

Remove a Table reference into the internal Table list. If the reference is not found, no modifications are made.

boolean remove\_table(unsigned)

Remove a Table reference into the internal Table list. If there is no Table reference in the indicated position, no modifications are made.

boolean remove(Condition\*)

Remove a Condition reference into the internal Condition list. If the reference is not found, no modifications are made.

boolean remove\_condition(unsigned)

Remove a Condition reference into the internal Condition list. If there is no Condition reference in the indicated position, no modifications are made.

boolean remove(Join\*)

Remove a Join reference into the internal Join list. If the reference is not found, no modifications are made.

boolean remove\_join(unsigned)

Remove a Join reference into the internal Join list. If there is no Join reference in the indicated position, no modifications are made.

const ConditionList\* conditions()

Returns a list of the Conditions that are defined for this Query.

const JoinList\* joins()

Returns a list of the Joins that are defined for this Query.

const TableList\* tables()

Returns a list of the Tables that are defined for this Query.

ReturnResult return\_result()

Returns information about the type of row results that will be returned by the query.

boolean evaluate()

This function constructs the SQL query that is defined by the objects that comprise this object. The generated SQL is then passed to Oracle to be parsed. Information about the success of this action is returned.

#### SEE ALSO

Condition(3N), Join(3N), Query (3N), SimpleQuery(3N), Table(3N)

### 3.1.3.9 DateColumn

#### NAME

DateColumn - wrapper class for Oracle DATE column datatype

#### SYNOPSIS

```
#include <nora/DateColumn.h>
```

#### DESCRIPTION

The DateColumn class provides an interface to Oracle columns defined as type "date." At run-time it is usually bound to specific named column and assumes information and data related to the named column.

#### CONSTRUCTORS

In almost all cases, objects of this type are created as a result of creating another object; i.e. Table or QueryResult. However, use of the Dual class provides the opportunity to directly instantiate this type of object.

```
DateColumn(Table* t, char* name, char* val, const char* in_fmt,  
const char* out_fmt )
```

Instantiation of a DateColumn object requires an associated table. The name parameter should match to the name of the database column and the (optional) value parameter initializes the contents. If a value is given it is required to be in the format, "DD-MON-YYYY" or it must specify the format used in the in\_fmt parameter. The default output format is specified as "YY MM DD", but may (optionally) specified in the out\_fmt parameter. In all cases, time as well as date information can be specified in a number of formats. See the Oracle manual for allowable date formatting options.

```
DateColumn(Table* t, DateColumn* cc)  
Copy constructor
```

#### MEMBER FUNCTIONS

```
const char* contents(boolean useCDF)
```

This function returns a string containing the name and value of the object. the useCDF parameter determines whether the output follows the CDF output standard. The default method is to use

CDF formatting.

`int oratype()`

Returns information about the type of data contained in the object. This is useful when working with Column objects.

`void obtain_date(const char*)`

Query database for date. The parameter is contains an SQL string asking for the date.

`unsigned second()`

Returns the number of seconds in the currently defined date.

`unsigned minute()`

Returns the number of minutes in the currently defined date.

`unsigned hour()`

Returns the number of hours in the currently defined date.

`unsigned day()`

Returns the day of the currently defined date. The numbers one through seven correspond to the days of the week, Sunday through Saturday.

`unsigned month()`

Returns the month of the currently defined date. The numbers one through twelve correspond to the months of the year, January through December.

`const char* month(unsigned m, boolean long_format)`

Returns a string containing the name of the month specified in the first parameter. The boolean parameter defines the format of the returned string. As the default (false), short months are returned (i.e. "Jan") instead of the full name (i.e. "January".) `unsigned year(boolean include_century)` Return the year of the currently defined date. The boolean parameter indicates whether to prefix the defined century.

`unsigned century()`

Return the century of the currently defined date.

`void today() ;`

Sets the defined date to be the current date and time.

`const char* output_format() ;`

Returns a string containing the format used to express the contained date.

`void output_format(const char*)`

Sets the format of the output date string. The format of the



parameter string should conform with the Oracle specification of a date string.

`const char* value()`

Returns a string containing the current value of this object.

`char* db_value()`

Used by other NORA objects for use with Oracle-specific operations.

`void* address()`

Used by other NORA objects for use with Oracle-specific operations.

`void operator= (const char*)`

Assignment operation used to set the value of the object. Care must be taken when using this method to ensure that the format of the date to be assigned equates to the input format defined in the object.

`void assign_value(const char*)`

Alternative means of assignment; identical to the `operator=` member function. Care must be taken when using this method to ensure that the format of the date to be assigned equates to the input format defined in the object.

`void assign_char_value(const char* val, const char*` Alternative means of assignment. Used to set the value of the object.

`unsigned length()`

Returns information about the length of the object's contents.

## SEE ALSO

Column (3N), Dual(3N), Table(3N), QueryResult(3N)

### 3.1.3.10 Expression

#### NAME

Expression - used by Condition class to define WHERE clause

#### SYNOPSIS

```
#include <nora/Expression.h>
```

#### DESCRIPTION

The Expression class is used to define singular relations within an SQL WHERE clause.

#### CONSTRUCTORS

Expression()

An empty constructor provides the basis for defining a relational clause. An empty constructor should not be passed to the Condition class.

Expression(Column\* col1, BooleanOp op, Column\* col2, boolean assign = false, boolean caseSensitive)

Construct relational clause that compares two table columns. The BooleanOp (op) parameter is a boolean relation that will be used to compare the two columns.

Expression(NumberColumn\* col1, BooleanOp op, int col2, boolean assign = false)

Construct relational clause that compares a Number-Column to a value. The BooleanOp (op) parameter is a boolean relation that will be used to compare the column to the value.

Expression(NumberColumn\* col1, BooleanOp, float col2, boolean assign = false)

Construct relational clause that compares a Number-Column to a value. The BooleanOp (op) parameter is a boolean relation that will be used to compare the column to the value.

Expression(NumberColumn\* col1, BooleanOp, const char\* col2, boolean assign = false)

Construct relational clause that compares a Number-Column to a string containing a value. The BooleanOp (op) parameter is a boolean relation that will be used to compare the column to the value.

Expression(FloatColumn\* col1, BooleanOp, const char\* col2,  
boolean assign = false)

Construct relational clause that compares a FloatColumn to a string containing a value. The BooleanOp (op) parameter is a boolean relation that will be used to compare the column to the value.

Expression(CharColumn\* col1, BooleanOp, const char\* col2,  
boolean assign = false, boolean caseSensitive = false)

Expression(DateColumn\* column, BooleanOp, const char\* date,  
boolean assign = false)

Construct relational clause that compares a CharColumn to a string value. The BooleanOp (op) parameter is a boolean relation that will be used to compare the column to the value.

## MEMBER FUNCTIONS

void reset()

Remove all existing relations.

void compare(Column\* col1, BooleanOp op, Column\* col2,  
boolean assign = false, boolean caseSensitive = false)

This function is used to prep the SQL segment in the case of a comparison between two columns.

void compare(NumberColumn\* col1, BooleanOp op, int col2,  
boolean assign = false)

This function is used to prep the SQL segment in the case of a comparison between a NumberColumn and a value.

void compare(NumberColumn\* col1, BooleanOp op, float col2,  
boolean assign = false)

This function is used to prep the SQL segment in the case of a comparison between a NumberColumn and a value.

void compare(NumberColumn\* col1, BooleanOp op, const char\* col2, boolean assign = false)

This function is used to prep the SQL segment in the case of a comparison between a NumberColumn and a value.

void compare(FloatColumn\* col1, BooleanOp, const char\* col2,  
boolean assign = false)

This function is used to prep the SQL segment in the case of a comparison between a FloatColumn and a value. void compare(CharColumn\* col1, BooleanOp op,

const char\* col2, boolean assign = false, boolean caseSensitive)

This function is used to prep the SQL segment in the case of a comparison between a CharColumn and a character string.

void compare(DateColumn\* col1, BooleanOp op, const char\* date, boolean assign = false)

This function is used to prep the SQL segment in the case of a comparison between a DateColumn and a date string. The date string must in the same format as the current input specification of the DateColumn.

const char\* statement()

Returns a character string that is used for inclusion in a complete SQL query.

void isnull(Column\* column)

Create SQL segment that tests whether the specified Column has a value.

void exists(Column\* column)

Create SQL segment that tests whether the specified Column has a value (is not null.)

## PROTECTED MEMBER FUNCTIONS

void build\_expr(const char\* col1, BooleanOp op, const char\* col2)

Construct SQL segment that defines relation.

## SEE ALSO

Condition (3N)

### 3.1.3.11 FetchedGroup

#### NAME

FetchGroup - Initiates database queries and handles ALL returned rows

#### SYNOPSIS

```
#include <nora/FetchedGroup.h>
```

#### DESCRIPTION

Unlike the FetchRows class, the FetchGroup returns all rows from a Query up to a defined limit. However, to be honest, this class has not yet been implemented.

#### CONSTRUCTORS

```
FetchGroup(Query*)
```

#### MEMBER FUNCTIONS

```
void fetch_value(unsigned col, char* value)
void fetch_value(unsigned col, unsigned* value)
void fetch_value(unsigned col, float* value)
void restart()
```

Cancel the outstanding query (if it exists) and restart.

```
unsigned count()
```

Returns the total number of returned rows in the query.

#### SEE ALSO

FetchRows (3N), Query(3N)

### 3.1.3.12 FetchedRows

#### NAME

FetchRows - Initiates database queries and handles returned rows

#### SYNOPSIS

```
#include <nora/FetchedRows.h>
```

#### DESCRIPTION

The FetchRows class provides the ability to iterate through the returned rows defined by a Query class.

#### CONSTRUCTORS

FetchRows(Query\*)

#### MEMBER FUNCTIONS

void fetch\_value(unsigned col, char\* value)  
Currently undefined.

void fetch\_value(unsigned col, unsigned\* value)  
Currently undefined.

void fetch\_value(unsigned col, float\* value)  
Currently undefined.

void restart()  
Cancel the outstanding query (if it exists) and restart.

int current()  
count to see where we are; negative if no more rows

unsigned next()  
Fetch the next row from the database. A negative value is returned if no more rows are available.

unsigned count()  
Returns the total number of returned rows in the query.

#### SEE ALSO

Query(3N)

### 3.1.3.13 FloatColumn

#### NAME

FloatColumn - wrapper class for Oracle FLOAT (NUMBER) column datatype

#### SYNOPSIS

```
#include <nora/FloatColumn.h>
```

#### DESCRIPTION

The FloatColumn class provides an interface to Oracle columns defined as type "float." At run-time it is usually bound to specific column and assumes information and data related to the column.

#### CONSTRUCTORS

In almost all cases, objects of this type are created as a result of creating another object; i.e. Table or QueryResult. However, use of the Dual class provides the opportunity to directly instantiate this type of object. Instantiation of a FloatColumn object requires an associated table. The table should have a column consistent with the type of this object (NUMBER.) FloatColumn(Table\* t, char\* name, float value) The name parameter should match the name of the database column and the (optional) value parameter initializes the object's contents.

FloatColumn(Table\* t, char\* name, double value)

The name parameter should match the name of the database column and the (optional) value parameter initializes the object's contents.

FloatColumn(Table\* t, FloatColumn\* fc)

Copy constructor.

#### MEMBER FUNCTIONS

const char\* contents(boolean)

This function returns a string containing the name and value of the object. The optional parameter determines whether the output follows the CDF standard and defaults to true if not given.

int oratype()

Returns information about the type of data contained in the object. This is useful when working with Column objects.

`const char* value()`

Returns a string containing the current value of this object.

`float float_value()`

Returns the value of the object's contents.

`double double_value()`

Returns the value of the object's contents.

`const char* db_value()`

Used by other NORA objects for use with Oracle-specific operations.

`void* address()`

Used by other NORA objects for use with Oracle-specific operations.

`void assign_value(const char*)`

Used to set the value of the object.

`void assign_value(float)`

Alternative means of assignment. Used to set the value of the object.

`void assign_value(double)`

Alternative means of assignment. Used to set the value of the object.

`void assign_char_value(const char*)`

Alternative means of assignment. Used to set the value of the object.

`unsigned length()`

Returns information about the length of the object's contents.

SEE ALSO

Column (3N)



### 3.1.3.14 ImmediateQuery

#### NAME

ImmediateQuery - query class that allows free-formatted queries

#### SYNOPSIS

```
#include <nora/ImmediateQuery.h>
```

#### DESCRIPTION

The ImmediateQuery is derived from the Query class and will accepted a valid SQL query that does not contain any wildcards in the SELECT list. The returned columns are stored in a QueryResult object. A restriction is in place that only allows SELECT statements to used.

#### CONSTRUCTORS

ImmediateQuery(QueryResult\* qr, const char\* query\_string)  
The constructor accepts a QueryResult reference to contain the resulting output of the query and an SQL SELECT statement that drives the query.

boolean next()  
During a query, fetches the next row corresponding to the query. Returns information about whether the fetch succeeded.

boolean evaluate()  
Trigger all of the contained objects to generate their SQL segents and send the result to Orace for parsing. Information about the success of the parsing operation is returned.

unsigned returned\_columns()  
Returns information about the number of columns that will be returned.

unsigned count()  
Returns the number of rows that will be returned.

boolean is\_query()  
Returns information about whether a valid SQL query is contained.

unsigned count\_columns()  
Returns information about the number of columns that will be

returned by the query.

#### SEE ALSO

FetchRows(3N), Query (3N), QueryResult (3N)

### 3.1.3.15 Join

#### NAME

Join - equivalent to Oracle joins; specify table relationships for queries

#### SYNOPSIS

```
#include <nora/Join .h>
```

#### DESCRIPTION

The Join class is used to specify table relationships for SQL queries. The relationship itself is realized in the WHERE clause of a generated SQL Query.

#### CONSTRUCTORS

Join(Column\* col\_a, Column\* col\_b, BooleanOp)

The constructor requires two Column objects and an (optional) third argument that defines the relationship between the two columns. If the third argument is not supplied, it defaults to "EQ"; an equality comparison.

#### MEMBER FUNCTIONS

void compare(BooleanOp)" The parameter modifies the relationship between the two Columns to use the specified relationship.

BooleanOp compare()" Returns information about the current relationship between the two Columns.

Column\* col\_a()" Returns a reference pointer to the first Column supplied in the constructor.

Column\* col\_b()" Returns a reference pointer to the second Column supplied in the constructor.

const char\* statement()" Returns a string containing the SQL code appropriate to the relationship to be used as part of a Query.

USAGE NOTES To improve query performance, it is necessary to consider the two tables corresponding to the two Columns specified in the constructor. The Column that is associated with the

table with the largest number of rows should be the first column specified in the constructor.

#### SEE ALSO

Column (3N), ComplexQuery (3N), SimpleQuery (3N)

### 3.1.3.16 LongColumn

#### NAME

LongColumn - wrapper class for Oracle LONG column datatype

#### SYNOPSIS

```
#include <nora/LongColumn.h>
```

#### DESCRIPTION

The LongColumn class provides an interface to Oracle columns defined as type "long." At run-time it is usually bound to specific column and assumes information and data related to the column.

#### CONSTRUCTORS

In almost all cases, objects of this type are created as a result of creating another object; i.e. Table or QueryResult. However, use of the Dual class provides the opportunity to directly instantiate this type of object.

LongColumn(Table\* t, unsigned length, char\* name, Instantiation of a LongColumn object requires an associated table. The length parameter determines the size of the object's contents and should correspond to the schema definition. The name parameter should also match the name of the database column and the (optional) value parameter initializes the object's contents.

LongColumn(Table\* t, LongColumn\* lc)" Copy constructor

#### MEMBER FUNCTIONS

const char\* contents(boolean)

This function returns a string containing the name and value of the object. The optional parameter determines whether the output follows the CDF standard and defaults to true if not given.

int oratype()

Returns information about the type of data contained in the object. This is useful when working with Column objects.

const char\* value()

Returns a string containing the current value of this object.

`const char* db_value()`

Used by other NORA objects for use with Oracle-specific operations.

`void* address()`

Used by other NORA objects for use with Oracle-specific operations. `void operator= (const char*)`

Assignment operation used to set the value of the object.

`void assign_value(const char*)`

Alternative means of assignment. Used to set the value of the object.

`void assign_char_value(const char*)`

Alternative means of assignment. Used to set the value of the object.

`unsigned length()`

Returns information about the length of the object's contents.

#### SEE ALSO

Column (3N), Dual (3N), Table(3N), QueryResult(3N)

### 3.1.3.17 QueryResult

NAME

QueryResult - pseduo-table used in conjunction with ImmediateQuery class

SYNOPSIS

```
#include <nora/QueryResult.h>
```

DESCRIPTION

The QueryResult class is derived from the Table class and is used to return

CONSTRUCTORS

QueryResult(char\*)

The single (optional) parameter "names" the object. This is currently not useful and defaults to "QueryResult".

MEMBER FUNCTIONS

const char\* contents(boolean)  
Generating CDF output is disabled.

boolean modified()  
This is an artifate of the Table implementation. Modifications to contained columns are not very useful.

boolean commit()  
This is an artifate of the Table implementation. Commits are not allowed.

void ignore\_all(boolean)  
This function is ignored since the SQL query is hardcoded; all columns are retrieved.

boolean ignore\_all()  
This function is ignored since the SQL query is hardcoded; all columns are retrieved.

void lock()  
Since no "actual" Table objects are being used, table locking is not allowed.

SEE ALSO

Column(3N), ImmediateQuery(3N), Table (3N)



### 3.1.3.18 RowID

#### NAME

RowID - wrapper class for Oracle ROWID column datatype

#### SYNOPSIS

```
#include <nora/RowID.h>
```

#### DESCRIPTION

The RowID class provides an interface to Oracle columns defined as type "rowid." At run-time it is usually bound to specific column and assumes information and data related to the column.

#### CONSTRUCTORS

In almost all cases, objects of this type are created as a result of creating another object; i.e. Table or QueryResult. However, use of the Dual class provides the opportunity to directly instantiate this type of object.

RowID(Table\* t)" Instantiation of a RowID object requires an associated table. There are no other parameters since the format is fixed.

#### MEMBER FUNCTIONS

int oratype()

Returns information about the type of data contained in the object. This is useful when working with Column objects.

const char\* value()

Returns a string containing the current value of this object.

const char\* db\_value()

Used by other NORA objects for use with Oracle-specific operations.

void\* address()

Used by other NORA objects for use with Oracle-specific operations.

unsigned length()

Returns information about the length of the objectUs contents.

## SEE ALSO

Column (3N), Dual (3N), Table(3N), QueryResult(3N)

### 3.1.3.19 Sequence

#### NAME

Sequence - interface to Oracle sequences

#### SYNOPSIS

```
#include <nora/Sequence.h>
```

#### DESCRIPTION

The Sequence classes is used to extract values from Oracle sequences. A sequence is a resource that returns a series of incremental/decremental values guaranteeing unique, successive values unless the sequence is configured to "roll over" and the cycle repeats.

#### CONSTRUCTORS

Sequence(char\* sequence\_name)

The parameter is a string containing the name of the Oracle sequence of interest.

#### MEMBER FUNCTIONS

char\* current\_value()

Returns a character string containing the current value of the sequence. Subsequent calls will return the same value.

char\* next\_value()

Retrieves the next value from the sequence. Subsequent calls will return new values constrained to the conditions described above.

### 3.1.3.20 SimpleQuery

#### NAME

SimpleQuery - basic interface for Oracle query capabilities

#### SYNOPSIS

```
#include <nora/SimpleQuery.h>
```

#### DESCRIPTION

The LongColumn class provides an interface to Oracle columns defined as type "long." At run-time it is usually bound to specific column and assumes information and data related to the column.

#### CONSTRUCTORS

SimpleQuery(Table\* t1, Condition\* c, Table\* t2, Join\* j, boolean uniq = false)

The constructor allows for the definition of a one table or two table relational join. The second Table parameter is optional as is the Join operator. A Join should be included if two Tables are specified.

SimpleQuery(Connection\* x, Table\* t1, Condition\* c, Table\* t2, Join\* j, boolean uniq = false)

Identical to the above constructor with the provision for specifying a Connection separate from the default Connection defined in the Database instance.

#### MEMBER FUNCTIONS

boolean next()

During a query, fetches the next row corresponding to the query. Returns information about whether the fetch succeeded.

void bind\_column(unsigned col, char\* val, unsigned len )

Bind a memory location to the column in the specified position. This should be used with CharColumn Column types.

void bind\_column(unsigned col, unsigned\* val)

Bind a memory location to the column in the specified position. This should be used with NumberColumn Column types.

void bind\_column(unsigned col, float\* val)

Bind a memory location to the column in the specified position. This should be used with FloatColumn Column types.

void bind\_column(Column\* col, char\* val)

Bind a memory location to the specified column. This should be used with CharColumn Column types.

void bind\_column(Column\* col, unsigned\* val)

Bind a memory location to the specified column. This should be used with NumberColumn Column types.

void bind\_column(Column\* col, float\* val)

Bind a memory location to the specified column. This should be used with FloatColumn Column types.

boolean evaluate()

Trigger all of the contained objects to generate their SQL segents and send the result to Orace for parsing. Information about the success of the parsing operation is returned.

unsigned returned\_columns()

Returns information about the number of columns that will be returned.

unsigned count()

Returns the number of rows that will be returned.

## PROTECTED MEMBER FUNCTIONS

const char\* table\_column\_names(Table\*)

Generates a string containing the names of the Tables used.

## SEE ALSO

Condition(3N), Join(3N), Query(3N), Table(3N)

### 3.1.3.21 Table

#### NAME

Table - wrapper for Oracle relational database tables

#### SYNOPSIS

```
#include <nora/Table.h>
```

#### DESCRIPTION

The Oracle Call Interface (OCI) routines do not allow binding to a table and accessing its functions, so the Table class is used to associate table relationships and table- >column relationships. Specifically, a Table object and its associated Column objects is similar to an Oracle table description; an Oracle table can be equated to a Table object.

#### CONSTRUCTORS

The Table class is derived from the DBObjct base class and is itself a generic base class for specifically constructed tables to be derived. The constructor is protected preventing any direct instantiation of this object. public:

const char\* name()

Returns string containing the name of the underlying Oracle table. The derived QueryResult class does not correspond to an Oracle table and will return the name used to instantiate the object.

unsigned columns()

Returns the number of columns associated with this table.

Column\* column(char\* column\_name)

This function will search its column list for a column with the given name. If a matching column name is found, a pointer to the Column class is returned. A null pointer is returned if no match is found.

Column\* column(unsigned position)

Returns a pointer to a Column class based upon its internal position. A null pointer if the supplied parameter is out of range.

void insert(unsigned position, Column\* col)

Insert a Column into the internal Table list at a specific position. The Column is appended to the list if the position exceed the

maximum position. This function is largely used by the code generators its use with generated code is not recommended.

`void append(Column* col)`

Append a Column to the end of the internal Table list . This function is largely used by the code generators its use with generated code is not recommended.

`void remove(unsigned position)`

Remove a Column at a specific position from the internal Table list. This function is largely used by the code generators its use with generated code is not recommended.

`void remove(char* column_name)`

Remove a Column, based upon the column names, from the internal Table list. If a Column with a matching name is not found, no changes are made. This function is largely used by the code generators its use with generated code is not recommended.

`const char* contents(boolean)`

This is a "pure" virtual function that returns a CDF-like formatted character string. This function must be defined by any derived class.

`boolean modified()`

Returns information about whether any of the Table's internal columns have been updated by user code.

`boolean remove_row()`

When used in conjunction with the Query classes, this function will "remove" a row from any subsequent queries.

`boolean commit()`

This function will "commit" the contents of the Table as though it were committing a record to the database. When used in conjunction with the Query classes, this function will update changes made to the Table. The successful completion of the operation is returned. Note that a commit is required at the Database level in order to make the changes permanent (unless the auto-commit feature is turned on.) `void ignore_all(boolean)` The parameter will determine whether the generated SQL code will retrieve all or none of the columns pertaining to this Table.

`boolean ignore_all()`

As a performance preventative measure, this function will modify the generated SQL code to NOT retrieve all columns pertaining to this Table.

`void lock()`

The function places an EXCLUSIVE LOCK on the associated

Oracle table until a commit() or rollback() is performed at the Database level.

## PROTECTED MEMBER FUNCTIONS

boolean row\_exists()

This is a "pure" virtual function and must be defined by any derived object. The function is during a table commit to determine if a record containing the same key is already in the database.

boolean update\_active\_row()

This is a "pure" virtual function and must be defined by any derived object. The function is used during a table commit to "audit" an existing row containing the same key to make way for a "new" row.

boolean remove\_active\_row()

This is a "pure" virtual function and must be defined by any derived object. The function alters a table row in a manner that effectively removes it from being returned by the SimpleQuery and ComplexQuery query classes.

## SEE ALSO

Column(3N)l, Database(3N), DBOject (3N), Query(3N)



### 3.1.3.22 RawColumn

#### NAME

RawColumn - wrapper class for Oracle LONG column datatype

#### SYNOPSIS

```
#include <nora/RawColumn.h>
```

#### DESCRIPTION

The RawColumn class provides an interface to Oracle columns defined as type "raw." At run-time it is usually bound to specific column and assumes information and data related to the column.

#### CONSTRUCTORS

In almost all cases, objects of this type are created as a result of creating another object; i.e. Table or QueryResult. However, use of the Dual class provides the opportunity to directly instantiate this type of object.

RawColumn(Table\* t, unsigned length, char\* name, Instantiation of a RawColumn object requires an associated table. The length parameter determines the size of the object's contents and should correspond to the schema definition. The name parameter should also match the name of the database column and the (optional) value parameter initializes the object's contents.

RawColumn(Table\* t, RawColumn\* rc)" Copy constructor

#### MEMBER FUNCTIONS

const char\* contents(boolean)

This function returns a string containing the name and value of the object. The optional parameter determines whether the output follows the CDF standard and defaults to true if not given.

int oratype()

Returns information about the type of data contained in the object. This is useful when working with Column objects.

const char\* value()

Returns a string containing the current value of this object.

`const char* db_value()`

Used by other NORA objects for use with Oracle-specific operations.

`void* address()`

Used by other NORA objects for use with Oracle-specific operations.

`void operator= (const char*)`

Assignment operation used to set the value of the object.

`void assign_value(const char*)`

Alternative means of assignment. Used to set the value of the object.

`void assign_char_value(const char*)`

Alternative means of assignment. Used to set the value of the object.

`unsigned length()`

Returns information about the length of the object's contents.

#### SEE ALSO

Column (3N), Dual (3N), Table(3N), QueryResult(3N)

### 3.1.3.23 Query

#### NAME

Query - abstract base class for Query classes

#### SYNOPSIS

```
#include <nora/Query.h>
```

#### DESCRIPTION

The Query class is a base class from which different types of query structures can be constructed. Derived classes have specific capabilities needed to initiate queries against an Oracle database.

#### CONSTRUCTORS

The constructor for the Query class is protected preventing direct instantiation. Derived classes must be careful to define the count() function which is the only "pure" virtual function.

#### MEMBER FUNCTIONS

```
int ora_error_val()
```

Returns the specific Oracle error code that was generated by the most recent error.

```
void bind_column(unsigned col, char* buf, unsigned bufl)
```

Bind the specified column from the select list to a memory location. The buf and bufl parameters indicate the address and size of the memory allocation.

```
void bind_column(unsigned col, unsigned* buf)
```

Bind the specified column from the select list to a memory location. The buf pointer refers to a memory address that would contain integer data (i.e. Number-Column.)

```
void bind_column(unsigned col, float* buf)
```

Bind the specified column from the select list to a memory location. The buf pointer refers to a memory address that would contain float data (i.e. FloatColumn.)

```
void bind_columns(Table* t, unsigned& query_pos)
```

Bind the columns in the specified Table based upon the position in the query (SELECT) statement.

void bind\_values(Table\* t, unsigned& query\_pos)

Bind the return values from an Oracle table fetch to the appropriate Table columns.

boolean next()

Fetch the next row corresponding to the query. Returns information about whether the fetch succeeded.

unsigned count()

Returns the number of rows that will be returned.

boolean evaluate()

Trigger all of the contained objects to generate their SQL segments and send the result to Oracle for parsing. Information about the success of the parsing operation is returned.

const char\* sql\_string()

Returns a string containing the generated SQL.

## PROTECTED MEMBER FUNCTIONS

void bind(unsigned col, void\* buf, unsigned buflen,

unsigned buftype = NULL\_STRING)

Low-level interface to Oracle bind routine.

unsigned count\_tokens(char\* token)

Returns information about the number of columns in the select list.

void execute()

Initiate routines to begin query.

void restart()

Cancel an outstanding query (if one exists) and restart.

void abort()

Cancel an outstanding query (if one exists.)

## SEE ALSO

DBObject (3N), Table (3N)

### 3.1.3.24 NumberColumn

NAME

NumberColumn - wrapper class for Oracle INTEGER (NUMBER) column datatype

SYNOPSIS

```
#include <nora/NumberColumn.h>
```

DESCRIPTION

The NumberColumn class provides an interface to Oracle columns defined as type "integer." At run-time it is usually bound to specific column and assumes information and data related to the column.

CONSTRUCTORS

In almost all cases, objects of this type are created as a result of creating another object; i.e. Table or QueryResult. However, use of the Dual class provides the opportunity to directly instantiate this type of object. Instantiation of a NumberColumn object requires an associated table. The table should have a column consistent with the type of this object (NUMBER.)

NumberColumn(Table\* t, char\* db\_col\_ref, short value)

The name parameter should match the name of the database column and the (optional) value parameter initializes the object's contents.

NumberColumn(Table\* t, char\* db\_col\_ref, int value)

The name parameter should match the name of the database column and the (optional) value parameter initializes the object's contents.

NumberColumn(Table\* t, char\* db\_col\_ref, long value)

The name parameter should match the name of the database column and the (optional) value parameter initializes the object's contents.

NumberColumn(Table\* t, NumberColumn\* nc)

Copy constructor

MEMBER FUNCTIONS

const char\* contents(boolean)

This function returns a string containing the name and value of the

object. The optional parameter determines whether the output follows the CDF standard and defaults to true if not given.

`int oratype()`

Returns information about the type of data contained in the object. This is useful when working with Column objects.

`const char* value()`

Returns a string containing the current value of this object.

`short short_value()`

Returns the value of the object's contents.

`int int_value()`

Returns the value of the object's contents.

`long long_value()`

Returns the value of the object's contents.

`const char* db_value()`

Used by other NORA objects for use with Oracle-specific operations.

`void* address()`

Used by other NORA objects for use with Oracle-specific operations.

`void assign_value(const char*)`

Used to set the value of the object.

`void assign_value(short)`

Alternative means of assignment. Used to set the value of the object.

`void assign_value(int)`

Alternative means of assignment. Used to set the value of the object.

`void assign_value(long)`

Alternative means of assignment. Used to set the value of the object.

`void assign_char_value(const char*)`

Alternative means of assignment. Used to set the value of the object.

`unsigned length()`

Returns information about the length of the object's contents.

SEE ALSO: Column (3N), Query (3N)

---

## 3.2 NARQ Library Principles

---

The NARQ library consists entirely of a suite of generated C++ code derived from a set of definitions describing the structure of the database schema. At present, each definition is the equivalent of a single database table, but this is only a matter of the GATEC implementation and does not represent a limitation of the library implementation. The additional benefits of this database-neutral arrangement is the ability to regenerate a complete interface for new database definitions as well as the ability to provide additional capabilities to selected objects by enhancing the C++ code generator.

In its current form, there is a three-step process in order to transition from static-file definition to generated C++ source; .FBI file -> database representation -> code generator. The net result is the NORA library and associated header files. The remainder of this section describes the specifics that characterize each step.

---

### 3.2.1 NARQ Library Generation

---

#### .FBI Files

The Field-Binding Interface (FBI) is the portable ASCII representation of a NARQ library object. The definition of an FBI file is similar to the structure of a C++ header file though different keywords are used. Figure 3-1 provides the current definition of an FBI file.

// Definitions required for object (optionally SQL) generation

Object <ObjectName>::[DatabaseTableName] {

Relationships:

// singular reference (one to one)

Object <reference\_object\_name>(<exported\_name>[,<exportedReference>]);

// multiple reference (one to many)

Objects <reference\_object\_name>(<exported\_name>[,<exported\_reference>]);

// NOTE: a single name within parentheses indicates that the names are

// identical for each object

// Derived relationship (sub-classing)

IsA <parent\_object\_name>(<member\_name>, <base\_member\_name>);

```

Exports:
    <exportedName> [ReadOnly]

Members:
    function:
        [virtual] <function_name>([<function_parameter>],{<function_parameter>});
    key:
        <member_name>::<table_name>.<column_name>
    data:
        <member_name>::<table_name>.<column_name>
};

```

*Figure 3-1. FBI definition.*

Figure 3-2 provides a sample of an existing file. In this case, the interface is the the Buyer database table (which will be used later in the Example code section.)

```

Object Buyer {
Relationships:
    Objects Acquisition(BuyerID, AssignedBuyer);
    Objects BuyerAssignment(BuyerID);
Exports:
    BuyerID
Members:
    key:
        number LocalSystemID::Buyer.LocalSystemID
        char BuyerID[BUYERID]::BuyerID.BuyerID
        char LastName[PER02]::BuyerID.LastName
        char FirstName[PER02]::BuyerID.FirstName
        char MiddleInitial[MINITIAL]::BuyerID.MiddleInitial
        char PhoneNumber[PER04]::BuyerID.PhoneNumber
        char EMailAddress[PER04]::BuyerID.EMailAddress
        char LeadStatus[BOOLEAN_VALUE]::BuyerID.LeadStatus
        char Download[BOOLEAN_VALUE]::BuyerID.Download
    }
}

```

*Figure 3-2. FBI definition of the Buyer object.*

In the above definition, it can quickly be discerned that the definition is broken into three distinct segments; Relationships, Exports and Members.

The Relationships section is a fairly application-specific area and identifies related objects, “Objects”, (or a singular object, “Object”). In the above example, it can be seen that a Buyer object can have a one-to-many relationship with an Acquisition object. Contained in parentheses is the basis of the relationship, or in SQL terms, the join relationship. The first name in parentheses is the name of the local object member name. The second name is the name of the external object’s point of relation. If the two names are



identical, it is not necessary. This is evidenced in the definition of the BuyerAssignment relationship. If multiple relationships exist between two objects, multiple definitions are allowed. An additional relationship, IsA, is also allowed. The IsA relationship allows for sub-classing relationships. The current GATEC implementation does not make use of this relationship.

The Exports section is used to effectively hide or limit the access of the key or data members defined in the third section. The current source generator has had this capability removed, but the definition has been retained. The design intent was to only generate read/write access functions to members defined in this section. The definition also allows for a "ReadOnly" qualifier that would signal the source generator to only provide read access functions for those members with the ReadOnly qualifier. Taken a step further, additional qualifiers could be added that allow only access to certain groups or classes of users.

The Members section can itself be separated into three separate segment. The first segment is the function segment and is entirely optional. The intent of the Function segment relates to the IsA relationship defined in the Relationships section. However, for the most part it is intended to provide a placeholder for specialized functions for an object that are inappropriate for the code generator. The key segment is required and must have at least one contained definition. The definitions in this section are almost exclusively distinguished as the key values upon which database indexes would be constructed. A definition in the key segment (and the data segment as well) is a mapping between the object and the database definition. (Refer to the data segment, following, for additional detail). As part of the code generation, the definition also provides some convenience function for both internal and programmatic use. The third segment, the data segment, is used to map the remaining database columns to the object. At a minimum, database columns which require values should be included. It is not necessary to include all database columns and is an effective way to exclude potentially sensitive columns from library access. The definition itself consists of two parts separated by a double colon. The first half is the local object definition; type, name and field size if a character type. The field size, in square brackets, can be given as a numeric value or as a predefined variable. These variables are defined elsewhere and are resolved when the database representation is generated. The second part of the definition is the database table name and column name to which the object is to be mapped. Part of the original design specified that the right hand side did not have to map to a database reference but could also resolve to a memory space for user applications. Nonetheless, this ability is missing from both the specification and the implementation.

The following sections describe each .FBI file used in the GATEC 2 system.

### 3.2.1.1 Acquisition Object

Object Acquisition {

Relationships:

Object Buyer(AssignedBuyer, BuyerID);

Object HoldStatus(HoldStatus, Status);

Objects BCASAward(UTNNumber);

Objects Document(UTNNumber);

Exports:

UTNNumber

HoldStatus

AssignedBuyer

RFQNumber

Members:

key:

char UTNNumber[UTNNUMBER]::Acquisition.UTNNumber

data:

char RFQNumber[RFQNUMBER]::Acquisition.RFQNumber

char SolicitationNumber[7]::Acquisition.SolicitationNumber

char SiteID[SITENUMBER]::Acquisition.SiteID

char DPASPriority[REF02]::Acquisition.DPASPriority

char InternalOrderNumber[REF02]::Acquisition.InternalOrderNumber

char PurchaseReqNumber[REF02]::Acquisition.PurchaseReqNumber

char AssignedBuyer[BUYERID]::Acquisition.AssignedBuyer

char HoldStatus[HOLD\_STATUS]::Acquisition.HoldStatus

dbDate HoldPeriod::Acquisition.HoldPeriod

char ReviewStatus[REVIEW\_STATUS]::Acquisition.ReviewStatus

char Priority[2]::Acquisition.Priority

}

### 3.2.1.2 Award Object

Object Award {

Relationships:

- Object AwardAcknowledgement(Acknowledgement, AcknowledgementType);
- Object OrganizationalEntity(BusEntityType, EntityIDCode);
- Object Currency(BuyerCurrencyCode, CurrencyCode);
- IsA Document(DocumentID);
- Object Contact(FirstContactID, ContactID);
- Object AwardPurchaseType(PurchaseType);
- Object Contact(SecondContactID, ContactID);
- Object Currency(SellerCurrencyCode, CurrencyCode);
- Object Contact(ThirdContactID, ContactID);

Exports:

- BuyerCurrencyCode
- SellerCurrencyCode
- BusEntityType
- ThirdContactID
- FirstContactID
- PurchaseType
- SecondContactID
- Acknowledgement
- DocumentID

Members:

key:

number DocumentID::Award.DocumentID

data:

- char PurchaseType[BEG02]::Award.PurchaseType
- char PurchaseOrderNumber[BEG03]::Award.PurchaseOrderNumber
- char CallDeliveryOrderNumber[BEG04]::Award.CallDeliveryOrderNumber
- dbDate EffectiveDate::Award.EffectiveDate
- char Acknowledgement[BEG07]::Award.Acknowledgement
- char AwardDescription[NTE02]::Award.AwardDescription
- char BuyerCurrencyCode[CUR02]::Award.BuyerCurrencyCode
- double BuyerExchangeRate::Award.BuyerExchangeRate
- dbDate BuyerRateEffective::Award.BuyerRateEffective
- dbDate BuyerRateExpires::Award.BuyerRateExpires
- char SellerCurrencyCode[CUR02]::Award.SellerCurrencyCode
- double SellerExchangeRate::Award.SellerExchangeRate
- dbDate SellerRateEffective::Award.SellerRateEffective
- dbDate SellerRateExpires::Award.SellerRateExpires
- char InternalOrderNumber[REF02]::Award.InternalOrderNumber
- char PurchaseReqNumber[REF02]::Award.PurchaseReqNumber
- char DPASPriority[REF02]::Award.DPASPriority
- char AcctgNAppropData[REF02]::Award.AcctgNAppropData
- char AcctgClassRefNumber[REF02]::Award.AcctgClassRefNumber
- char QuoteReferenceNumber[REF02]::Award.QuoteReferenceNumber
- dbDate QuoteReferenceDate::Award.QuoteReferenceDate
- char RFQReferenceNumber[BQR02]::Award.RFQReferenceNumber
- dbDate RFQReferenceDate::Award.RFQReferenceDate

dbDate RequiredDeliveryDate::Award.RequiredDeliveryDate  
char BusEntityType[N101]::Award.BusEntityType  
char BusEntityName[N102]::Award.BusEntityName  
number BusEntityVendorID::Award.BusEntityVendorID  
char BusEntityDept[N201]::Award.BusEntityDept  
char BusEntityAddress[N301]::Award.BusEntityAddress  
char BusEntityCity[N401]::Award.BusEntityCity  
char BusEntityState[N402]::Award.BusEntityState  
char BusEntityZIP[N403]::Award.BusEntityZIP  
char BidNumber[REF02]::Award.BidNumber  
char BuyersOfficeSymbol[REF02]::Award.BuyersOfficeSymbol  
char CriticalityDesignator[REF02]::Award.CriticalityDesignator  
char FirstContactID[CONTACTID]::Award.FirstContactID  
char SecondContactID[CONTACTID]::Award.SecondContactID  
char ThirdContactID[CONTACTID]::Award.ThirdContactID

}

### 3.2.1.3 AwardLineItem Object

Object AwardLineItem {

Relationships:

- Object Document(DocumentID);
- Object FederalStockClass(FedStockClass, FedStockClassID);
- IsA LineItem(ItemNumber);
- Object UnitOfMeasure(UnitOfMeasure, UnitOfMeasureCode);
- Object UnitPriceCodeBasis(UnitPriceBasis);

Exports:

- FedStockClass
- ItemNumber
- UnitPriceBasis
- UnitOfMeasure

Members:

key:

- number DocumentID::AwardLineItem.DocumentID
- char ItemNumber[PO101]::AwardLineItem.ItemNumber

data:

- char DPASPriority[REF02]::AwardLineItem.DPASPriority
- char InternalOrderNumber[REF02]::AwardLineItem.InternalOrderNumber
- char PurchaseReqNumber[REF02]::AwardLineItem.PurchaseReqNumber
- short SingleDeliveryDate::AwardLineItem.SingleDeliveryDate
- dbDate DeliveryDate::AwardLineItem.DeliveryDate
- double TotalLineAmount::AwardLineItem.TotalLineAmount
- double Quantity::AwardLineItem.Quantity
- char UnitOfMeasure[PO103]::AwardLineItem.UnitOfMeasure
- double UnitPrice::AwardLineItem.UnitPrice
- char UnitPriceBasis[PO105]::AwardLineItem.UnitPriceBasis
- char FedStockClass[4]::AwardLineItem.FedStockClass
- char StdIndustrialClass[PO109]::AwardLineItem.StdIndustrialClass
- char PartListIncluded[1]::AwardLineItem.PartListIncluded
- char VariationPercent[2]::AwardLineItem.VariationPercent
- char PurchaseVariation[1]::AwardLineItem.PurchaseVariation
- char BuyerName[N102]::AwardLineItem.BuyerName
- char BuyerCageCode[N104]::AwardLineItem.BuyerCageCode
- char BuyerDept[N201]::AwardLineItem.BuyerDept
- char BuyerAddress[N301]::AwardLineItem.BuyerAddress
- char BuyerCity[25]::AwardLineItem.BuyerCity
- char BuyerState[N402]::AwardLineItem.BuyerState
- char BuyerZIP[N403]::AwardLineItem.BuyerZIP
- char ShipToName[N102]::AwardLineItem.ShipToName
- number ShipToVendorID::AwardLineItem.ShipToVendorID
- char ShipToDept[N201]::AwardLineItem.ShipToDept
- char ShipToAddress[N301]::AwardLineItem.ShipToAddress
- char ShipToCity[25]::AwardLineItem.ShipToCity
- char ShipToState[N402]::AwardLineItem.ShipToState
- char ShipToZIP[N403]::AwardLineItem.ShipToZIP
- char BillToName[N102]::AwardLineItem.BillToName
- number BillToVendorID::AwardLineItem.BillToVendorID

char BillToDept[N201]::AwardLineItem.BillToDept  
char BillToAddress[N301]::AwardLineItem.BillToAddress  
char BillToCity[25]::AwardLineItem.BillToCity  
char BillToState[N402]::AwardLineItem.BillToState  
char BillToZIP[N403]::AwardLineItem.BillToZIP

### 3.2.1.4 AwardPurchaseType Object

```
Object AwardPurchaseType {  
  Relationships:  
    Objects Award(PurchaseType);  
  Exports:  
    PurchaseType  
  Members:  
    key:      char PurchaseType[BEG02]::AwardPurchaseType.PurchaseType  
    data:  
}
```

### 3.2.1.5 BCASAward Object

Object BCASAward {

Relationships:

Object CompetitionCode(CompetitionCode, Competitive);  
Object NegotiationAuthority(NegotiationAuthority, Authority);  
Object Acquisition(UTNNumber);

Exports:

UTNNumber  
CompetitionCode  
NegotiationAuthority

Members:

key:

char UTNNumber[UTNNUMBER]::BCASAward.UTNNumber

data:

char VendorCode[VENDOR\_CODE]::BCASAward.VendorCode  
char NegotiationAuthority[NEGO\_AUTH]::BCASAward.NegotiationAuthority  
char CompetitionCode[COMP\_CODE]::BCASAward.CompetitionCode  
char SolicitationNumber[7]::BCASAward.SolicitationNumber  
char PIIN[PIIN\_LEN]::BCASAward.PIIN  
char OrderStatements[30]::BCASAward.OrderStatements  
char ConfirmWith[15]::BCASAward.ConfirmWith  
char ContractRefNumber[30]::BCASAward.ContractRefNumber

}



### 3.2.1.6 Buyer Object

Object Buyer {

Relationships:

Objects Acquisition(BuyerID, AssignedBuyer);

Objects BuyerAssignment(BuyerID);

Exports:

BuyerID

Members:

key:

number LocalSystemID::Buyer.LocalSystemID

data:

char BuyerID[BUYERID]::Buyer.BuyerID

char LastName[PER02]::Buyer.LastName

char FirstName[PER02]::Buyer.FirstName

char MiddleInitial[MINITIAL]::Buyer.MiddleInitial

char PhoneNumber[PER04]::Buyer.PhoneNumber

char EMailAddress[PER04]::Buyer.EMailAddress

char LeadStatus[BOOLEAN\_VALUE]::Buyer.LeadStatus

char Download[BOOLEAN\_VALUE]::Buyer.Download

}

### 3.2.1.7 BuyerAssignment Object

Object BuyerAssignment {

Relationships:

Object Buyer(BuyerID);

Object FederalStockClass(FedStockClass, FedStockClassID);

Exports:

BuyerID

FedStockClass

Members:

key:

char BuyerID[BUYERID]::BuyerAssignment.BuyerID

data:

char FedStockClass[4]::BuyerAssignment.FedStockClass

}

### 3.2.1.8 BuyerNote Object

```
Object BuyerNote {
Relationships:
    Object Vendor(VendorID, VendorID);
    Object Document(DocumentID, DocumentID);
Exports:
    DocumentID
    VendorID
Members:
    key:
        number DocumentID::BuyerNote.DocumentID
    data:
        long NoteNumber::BuyerNote.NoteNumber
        char Note[MAX_TEXT]::BuyerNote.Note
        number VendorID::BuyerNote.VendorID
        dbDate InclusionDate::BuyerNote.InclusionDate
}
```

### 3.2.1.9 CancellationCode Object

Object CancellationCode {

Relationships:

Objects SolicitationHistory(CancelCode, CancellationCode);

Exports:

CancelCode

Members:

key:

char CancelCode[CNX\_CODE]::CancellationCode.CancelCode

data:

}

### 3.2.1.10 Clause

Object Clause {

Relationships:

Object ClauseCertification(ClaueCertification, RefNumQualifier);  
IsA RelatedPaperwork(PaperworkID);

Exports:

PaperworkID  
ClauseCertification

Members:

key:

number PaperworkID::Clause.PaperworkID

data:

char ClauseCertification[N901]::Clause.ClauseCertification  
char ClauseRefNumber[N902]::Clause.ClauseRefNumber  
char ClauseSource[N903]::Clause.ClauseSource  
char ClauseExplanation[N903]::Clause.ClauseExplanation

}

### 3.2.1.11 ClauseCertification

Object ClauseCertification {

Relationships:

Objects Clause(RefNumQualifier, ClauseCertification);

Exports:

RefNumQualifier

Members:

key:

char RefNumQualifier[N901]::ClauseCertification.RefNumQualifier

data:

}

### 3.2.1.12 Commuicator Object

Object Communicator {

Relationships:

Object Contact(FirstContactID, ContactID);  
Object Contact(SecondContactID, ContactID);  
Object Contact(ThirdContactID, ContactID);

Exports:

ThirdContactID  
CommunicatorID  
SecondContactID  
FirstContactID

Members:

key:

long CommunicatorID::Communicator.CommunicatorID

data:

char LastName[N201]::Communicator.LastName  
char FirstName[N102]::Communicator.FirstName  
char Address[N301]::Communicator.Address  
char City[N401]::Communicator.City  
char State[N402]::Communicator.State  
char ZIP[N403]::Communicator.ZIP  
char FirstContactID[CONTACTID]::Communicator.FirstContactID  
char SecondContactID[CONTACTID]::Communicator.SecondContactID  
char ThirdContactID[CONTACTID]::Communicator.ThirdContactID

}

### 3.2.1.13 CompetitionCode Object

Object CompetitionCode {

Relationships:

Objects BCASAAward(Competitive, CompetitionCode);

Objects SolicitationHistory(Competitive, CompetitionCode);

Exports:

Competitive

Members:

key:

char Competitive[COMP\_CODE]::CompetitionCode.Competitive

data:

}



### 3.2.1.14 Contact Object

```
Object Contact {
Relationships:
    Object PreferredAccess(PreferredAccess, CommNumQual);
Exports:
    ContactID
    PreferredAccess
Members:
    key:
        char ContactID[CONTACTID]::Contact.ContactID
    data:
        char Name[PER02]::Contact.Name
        char PreferredAccess[PER03]::Contact.PreferredAccess
        char PhoneNumber[PER04]::Contact.PhoneNumber
        char FaxNumber[PER04]::Contact.FaxNumber
        char EMailAddress[PER04]::Contact.EMailAddress
}
```

### 3.2.1.15 ControlStandards Object

Object ControlStandards {  
Relationships:  
    Object InterchangeControlHdr(Standard, InterchangeCtlStds)  
Exports:  
Members:  
    key:  
        char Standard[I10]::ControlStandards.ControlStandard  
    data:  
        char Definition[255]::ControlStandards.Definition  
}

### 3.2.1.16 ControlVersion Object

Object ControlVersion {  
Relationships:  
    Object InterchangeControlHdr(VersionNumber, InterchangeVersion)  
Exports:  
Members:  
    key:  
        char VersionNumber[I11]::ControlVersion.ControlVersion  
    data:  
        char Definition[255]::ControlVersion.Definition  
        char DMNumber[8]::ControlVersion.DMNumber  
}

### 3.2.1.17 Currency Object

Object Currency {  
Relationships:  
    Objects SolicitationHistory(BCASCurrency, Currency);  
    Objects Award(CurrencyCode, BuyerCurrencyCode);  
    Objects Award(CurrencyCode, SellerCurrencyCode);  
    Objects Quote(CurrencyCode);  
Exports:  
    CurrencyCode  
    BCASCurrency  
Members:  
    key:  
        char CurrencyCode[CUR02]::Currency.CurrencyCode  
    data:  
        char BCASCurrency[CUR\_CODE]::Currency.BCASCurrency  
}

### 3.2.1.18 DeliverySchedule Object

```
Object DeliverySchedule {
Relationships:
    IsA RelatedPaperwork(PaperworkID);
    Object UnitOfMeasure(UnitOfMeasure, UnitOfMeasureCode);
Exports:
    PaperworkID
    UnitOfMeasure
Members:
    key:
        number PaperworkID::DeliverySchedule.PaperworkID
    data:
        char LineItemNumber[4]::DeliverySchedule.LineItemNumber
        double ScheduledQuantity::DeliverySchedule.ScheduledQuantity
        char UnitOfMeasure[SCH02]::DeliverySchedule.UnitOfMeasure
        dbDate ScheduledDate::DeliverySchedule.ScheduledDate
}
```

### 3.2.1.19 Document Object

Object Document {

Relationships:

- Objects BuyerNote(DocumentID);
- Objects DocumentAddressee(DocumentID);
- Objects LineItem(DocumentID);
- Objects RelatedPaperwork(DocumentID);
- Object DocumentStatus(DocumentStatus, Status);
- Object DocumentType(DocumentType, TransactionSetID);
- Object DocumentVersion(DocumentVersion, VersionID);
- Object ReviewStatus(ReviewStatus, Status);
- Object Acquisition(UTNNumber);

Exports:

- UTNNumber
- DocumentType
- ReviewStatus
- DocumentVersion
- DocumentStatus
- DocumentID

Members:

key:

number DocumentID::Document.DocumentID

data:

- char UTNNumber[UTNNUMBER]::Document.UTNNumber
- char TransactionNumber[TRANSACTION]::Document.TransactionNumber
- char DocumentType[ST01]::Document.DocumentType
- char DocumentVersion[VERSION\_ID]::Document.DocumentVersion
- char X12ReferenceNumber[TRANSACTION]::Document.X12ReferenceNumber
- dbDate EffectiveDate::Document.EffectiveDate
- dbDate ExpirationDate::Document.ExpirationDate
- char DocumentStatus[EL\_353]::Document.DocumentStatus
- char ReviewStatus[REVIEW\_STATUS]::Document.ReviewStatus

}

### 3.2.1.20 DocumentAddressee Object

```
Object DocumentAddressee {
Relationships:
    Object Vendor(VendorID);
    Object Document(DocumentID);
Exports:
    DocumentID
    VendorID
Members:
    key:
        number DocumentID::DocumentAddressee.DocumentID
        number VendorID::DocumentAddressee.VendorID
    data:
        dbDate TransmittalDate::DocumentAddressee.TransmittalDate
}
```

### 3.2.1.23 DocumentType Object

Object DocumentType {

Relationships:

Objects Document(TransactionSetID, DocumentType);

Objects LineItem(TransactionSetID, DocumentType);

Exports:

TransactionSetID

Members:

key:

char TransactionSetID[ST01]::DocumentType.TransactionSetID

data:

char TypeDescription[MAX\_TEXT]::DocumentType.TypeDescription

}

### 3.2.1.24 DocumentVersion Object

Object DocumentVersion {

Relationships:

    Objects Document(VersionID, DocumentVersion);

    Object DocumentVersionType(VersionType);

Exports:

    VersionID

    VersionType

Members:

    key:

        char VersionID[VERSION\_ID]::DocumentVersion.VersionID

    data:

        char VersionType[MAX\_CODE]::DocumentVersion.VersionType

        dbDate VersionDate::DocumentVersion.VersionDate

}

### 3.2.1.25 DocumentVersionType Object

Object DocumentVersionType {

Relationships:

Objects DocumentVersion(VersionType);

Exports:

VersionType

Members:

key:

char VersionType[MAX\_CODE]::DocumentVersionType.VersionType

data:

char VersionDescription[MAX\_TEXT]::DocumentVersionType.VersionDescription

}



### 3.2.1.26 DownloadStockClass Object

```
Object DownloadStockClass {  
  Relationships:  
  Exports:  
    FedStockClass  
  Members:  
    key:  
      char FedStockClass[5]::DownloadStockClass.FedStockClass  
}
```

### 3.2.1.27 FOBAcceptancePoint Object

```
Object FOBAcceptancePoint {  
  Relationships:  
    Objects FreeOnBoard(LocationQual, FOBAcceptancePoint);  
    Objects FreeOnBoard(LocationQual, FOBType);  
  Exports:  
    LocationQual  
  Members:  
    key:  
      char LocationQual[EL_309]::FOBAcceptancePoint.LocationQual  
    data:  
}
```

### 3.2.1.28 FederalStockClass Object

Object FederalStockClass {

Relationships:

- Objects AwardLineItem(FedStockClassID, FedStockClass);
- Objects BuyerAssignment(FedStockClassID, FedStockClass);
- Objects LineItem(FedStockClassID, FedStockClass);
- Objects QuoteLineItem(FedStockClassID, FedStockClass);
- Objects ReqForQuoteLineItem(FedStockClassID, FedStockClass);

Exports:

FedStockClassID

Members:

key:

char FedStockClassID[4]::FederalStockClass.FedStockClassID

char Suffix[3]::FederalStockClass.Suffix

data:

char ClassDescription[MAX\_TEXT]::FederalStockClass.ClassDescription

}

### 3.2.1.29 FreeOnBoard Object

```
Object FreeOnBoard {
Relationships:
    Object FOBAcceptancePoint(FOBAcceptancePoint, LocationQual);
    Object FOBAcceptancePoint(FOBType, LocationQual);
    IsA RelatedPaperwork(PaperworkID);
Exports:
    FOBAcceptancePoint
    FOBType
    PaperworkID
Members:
    key:
        number PaperworkID::FreeOnBoard.PaperworkID
    data:
        char FOBType[FOB02]::FreeOnBoard.FOBType
        char FOBDescription[FOB03]::FreeOnBoard.FOBDescription
        char FOBAcceptancePoint[FOB06]::FreeOnBoard.FOBAcceptancePoint
        char
FOBAlternateInspection[BOOLEAN_VALUE]::FreeOnBoard.FOBAlternateInspection
        char FOBInspectionPoint[FOB07]::FreeOnBoard.FOBInspectionPoint
}
```

### 3.2.1.30 FunctionalAck Object

```
Object FunctionalAck {
Relationships:
    IsA Document(DocumentID);
Exports:
    DocumentID
Members:
    key:
        number DocumentID::FunctionalAck.DocumentID
    data:
        number GroupControlNumber::FunctionalAck.GroupControlNumber
        char FunctionalGroupAckCode[1]::FunctionalAck.FunctionalGroupAckCode
}
```

### 3.2.1.31 FunctionalGroupHdr Object

```
Object FunctionalGroupHdr {
Relationships:
Exports:
    CageCode
Members:
    key:
        char CageCode[N104]::FunctionalGroupHdr.CageCode
    data:
        char ApplicationSendersCode[15]::FunctionalGroupHdr.ApplicationSendersCode
        char ApplicationReceiversCode[15]::FunctionalGroupHdr.ApplicationReceiversCode
        dbDate GroupDate::FunctionalGroupHdr.GroupDate
        dbTime GroupTime::FunctionalGroupHdr.GroupTime
        char GroupControlNumber[9]::FunctionalGroupHdr.GroupControlNumber
        char ResponsibleAgencyCode[2]::FunctionalGroupHdr.ResponsibleAgencyCode
        char VersionReleaseCode[12]::FunctionalGroupHdr.VersionReleaseCode
}
```

### 3.2.1.32 FundCode Object

```
Object FundCode {
Relationships:
Exports:
    Fund
Members:
    key:
        char Fund[FUND_CODE]::FundCode.Fund
    data:
        double DiscretionPercentage::FundCode.DiscretionPercentage
}
```

### 3.2.1.33 GSDefaults Object

Object GSDefaults {

Relationships:

Exports:

Members:

key:

char DocumentType[4]::GSDefaults.DocumentType

data:

char FunctionalID[2]::GSDefaults.FunctionalID

char ApplicationSender[15]::GSDefaults.ApplicationSender

char ApplicationReceiver[15]::GSDefaults.ApplicationReceiver

dbDate GroupDate::GSDefaults.GroupDate

dbDate GroupTime::GSDefaults.GroupTime

long GroupControlNumber::GSDefaults.GroupControlNumber

char ResponsibleAgency[2]::GSDefaults.ResponsibleAgency

char InterchangeVersion[12]::GSDefaults.InterchangeVersion

}

### 3.2.1.34 HoldStatus Object

```
Object HoldStatus {  
  Relationships:  
    Objects Acquisition(Status, HoldStatus);  
  Exports:  
    Status  
  Members:  
    key:  
      char Status[HOLD_STATUS]::HoldStatus.Status  
    data:  
}  

```

### 3.2.1.35 Holidays Object

```
Object Holidays {  
  Relationships:  
  Exports:  
  Members:  
    key:      dbDate Holiday::Holidays.Holiday  
    data:     char Description[255]::Holidays.Description  
}
```

### 3.2.1.36 ISAAuthQualifier Object

Object ISAAuthQualifier {

Relationships:

Object InterchangeControlHdr(AuthQualifier, AuthorizationID)

Exports:

AuthQualifier

Members:

key:

char AuthQualifier[I01]::ISAAuthQualifier.AuthQualifier

data:

char Definition[255]::ISAAuthQualifier.Definition

}

### 3.2.1.37 ISADefaults Object

Object ISADefaults {

Relationships:

Exports:

Members:

key:

char DocumentType[4]::ISADefaults.DocumentType

data:

char AuthorizationID[2]::ISADefaults.AuthorizationID

char Authorization[10]::ISADefaults.Authorization

char SecurityID[2]::ISADefaults.SecurityID

char Security[10]::ISADefaults.Security

char SenderIDQualifier[2]::ISADefaults.SenderIDQualifier

char SenderID[15]::ISADefaults.SenderID

char ReceiverIDQualifier[2]::ISADefaults.ReceiverIDQualifier

char ReceiverID[15]::ISADefaults.ReceiverID

dbDate InterchangeDate::ISADefaults.InterchangeDate

dbTime InterchangeTime::ISADefaults.InterchangeTime

char InterchangeCtlStds[1]::ISADefaults.InterchangeCtlStds

char InterchangeVersion[5]::ISADefaults.InterchangeVersion

long ControlNumber::ISADefaults.ControlNumber

char AckRequested[1]::ISADefaults.AckRequested

char TestIndicator[1]::ISADefaults.TestIndicator

char SubElementSeparator[1]::ISADefaults.SubElementSeparator

char ElementSeparator[1]::ISADefaults.ElementSeparator

}



### 3.2.1.38 ISAInterchangeQualifier Object

Object ISAInterchangeQualifier {

Relationships:

Object InterchangeControlHdr(InterchangeQualifier, InterchangeID)

Exports:

Members:

key:

char InterchangeQualifier[I05]::ISAInterchangeQualifier.InterchangeQualifier

data:

char Definition[255]::ISAInterchangeQualifier.Definition

}

### 3.2.1.39 InterchangeControlHdr Object

Object InterchangeControlHdr {

Relationships:

- Object ISAAuthQualifier(AuthorizationID, AuthQualifier)
- Object ControlStandards(InterchangeCtlStds, Standard)
- Object ControlVersion(InterchangeVersion, VersionNumber)
- Object ISAInterchangeQualifier(InterchangeID, InterchangeQualifier)
- Object InterchangeRecipient(SenderID, RecipientID)
- Object InterchangeRecipient(ReceiverID, RecipientID)

Exports:

Members:

key:

char CageCode[N104]::InterchangeControlHdr.CageCode

data:

char SenderID[I06]::InterchangeControlHdr.SenderID  
long ControlNumber::InterchangeControlHdr.ControlNumber  
char AuthorizationID[I01]::InterchangeControlHdr.AuthorizationID  
char Authorization[I02]::InterchangeControlHdr.Authorization  
char SecurityID[I03]::InterchangeControlHdr.SecurityID  
char Security[I04]::InterchangeControlHdr.Security  
char InterchangeID[I05]::InterchangeControlHdr.InterchangeID  
char ReceiverID[I07]::InterchangeControlHdr.ReceiverID  
dbDate InterchangeDate::InterchangeControlHdr.InterchangeDate  
dbTime InterchangeTime::InterchangeControlHdr.InterchangeTime  
char InterchangeCtlStds[I10]::InterchangeControlHdr.InterchangeCtlStds  
char InterchangeVersion[I11]::InterchangeControlHdr.InterchangeVersion  
char AckRequested[I13]::InterchangeControlHdr.AckRequested  
char TestIndicator[I14]::InterchangeControlHdr.TestIndicator  
char SubElementSeparator[I15]::InterchangeControlHdr.SubElementSeparator  
int FunctionalGroups::InterchangeControlHdr.FunctionalGroups  
char AckCode[I17]::InterchangeControlHdr.AckCode  
char NoteCode[I18]::InterchangeControlHdr.NoteCode  
char ElementSeparator[1]::InterchangeControlHdr.ElementSeparator

}

### 3.2.1.40 InterchangeRecipient Object

```
Object InterchangeRecipient {
Relationships:
Exports:
    RecipientID
Members:
    key:
        char RecipientID[I07]::InterchangeRecipient.RecipientID
    data:
        char RecipientName[255]::InterchangeRecipient.RecipientName
}
```

### 3.2.1.41 InvoiceAddress Object

```
Object InvoiceAddress {
Relationships:
Exports:
Members:
    key:
        char EntityIDCode[N101]::InvoiceAddress.EntityIDCode
    data:
        char Name[N102]::InvoiceAddress.Name
        char IDCodeQualifier[N103]::InvoiceAddress.IDCodeQualifier
        char IDCode[N104]::InvoiceAddress.IDCode
        char Department[N201]::InvoiceAddress.Department
        char Address[N301]::InvoiceAddress.Address
        char City[N401]::InvoiceAddress.City
        char State[N402]::InvoiceAddress.State
        char ZIP[N403]::InvoiceAddress.ZIP
        char Country[N404]::InvoiceAddress.Country
}
```

### 3.2.1.42 Item Object

Object Item {

Members:

key:

char StockNumber[15]::Item.StockNumber

data:

char SupNomenInd[1]::Item.SupNomenInd

char ContrInd[1]::Item.ContrInd

char Suffix[2]::Item.Suffix

char UnitOfIssue[2]::Item.UnitOfIssue

char BuyerCode[3]::Item.BuyerCode

char CustomerID[1]::Item.CustomerID

char VariationInQuantity[2]::Item.VariationInQuantity

char AutomaticPurchaseOrder[2]::Item.AutomaticPurchaseOrder

char BrandNameOrSoleSource[2]::Item.BrandNameOrSoleSource

char RecDate[5]::Item.RecDate

char CommodityAssignment[1]::Item.CommodityAssignment

char DateLastAward[5]::Item.DateLastAward

char ManufacturerName[30]::Item.ManufacturerName

char ManufacturerPart[20]::Item.ManufacturerPart

char Nomenclature01[40]::Item.Nomenclature01

char Nomenclature02[40]::Item.Nomenclature02

char Nomenclature03[40]::Item.Nomenclature03

char Nomenclature04[40]::Item.Nomenclature04

char Nomenclature05[40]::Item.Nomenclature05

char Nomenclature06[40]::Item.Nomenclature06

}

### 3.2.1.43 ItemDetails Object

Object ItemDetails {

Relationships:

- Object UnitOfMeasure(LinearUnitOfMeasure, UnitOfMeasureCode);
- Object ItemPackageType(PackageType, PackagingCode);
- IsA RelatedPaperwork(PaperworkID);
- Object UnitOfMeasure(SizeUnitOfMeasure, UnitOfMeasureCode);
- Object UnitOfMeasure(VolumeUnitOfMeasure, UnitOfMeasureCode);
- Object ItemWeightType(WeightType, WeightQual);
- Object UnitOfMeasure(WeightUnitOfMeasure, UnitOfMeasureCode);

Exports:

- WeightUnitOfMeasure
- WeightType
- SizeUnitOfMeasure
- PackageType
- PaperworkID
- LinearUnitOfMeasure
- VolumeUnitOfMeasure

Members:

key:

number PaperworkID::ItemDetails.PaperworkID

data:

- long ItemsPerUnit::ItemDetails.ItemsPerUnit
- double InnerPackSize::ItemDetails.InnerPackSize
- char SizeUnitOfMeasure[PO403]::ItemDetails.SizeUnitOfMeasure
- char PackageType[PO404]::ItemDetails.PackageType
- char WeightType[PO405]::ItemDetails.WeightType
- double PackageWeight::ItemDetails.PackageWeight
- char WeightUnitOfMeasure[PO407]::ItemDetails.WeightUnitOfMeasure
- double GrossVolumePerPack::ItemDetails.GrossVolumePerPack
- char VolumeUnitOfMeasure[PO409]::ItemDetails.VolumeUnitOfMeasure
- double PackageLength::ItemDetails.PackageLength
- double PackageWidth::ItemDetails.PackageWidth
- double PackageHeight::ItemDetails.PackageHeight
- char LinearUnitOfMeasure[PO413]::ItemDetails.LinearUnitOfMeasure

}

### 3.2.1.44 ItemPackageType Object

Object ItemPackageType {

Relationships:

Objects ItemDetails(PackagingCode, PackageType);

Exports:

PackagingCode

Members:

key:

char PackagingCode[PO404]::ItemPackageType.PackagingCode

data:

}

### 3.2.1.45 ItemWeightType Object

Object ItemWeightType {

Relationships:

Objects ItemDetails(WeightQual, WeightType);

Exports:

WeightQual

Members:

key:

char WeightQual[PO405]::ItemWeightType.WeightQual

data:

}

### 3.2.1.46 LineItem Object

Object LineItem {

Relationships:

- Object Document(DocumentID);
- Objects Part(DocumentID);
- Object DocumentType(DocumentType, TransactionSetID);
- Object FederalStockClass(FedStockClass, FedStockClassID);
- Objects Part(ItemNumber);
- Objects RelatedPaperwork(ItemNumber, LineItemNumber);
- Object LineItemStatus(StatusCode);

Exports:

- StatusCode
- FedStockClass
- DocumentType
- ItemNumber
- DocumentID

Members:

key:

- number DocumentID::LineItem.DocumentID
- char ItemNumber[PO101]::LineItem.ItemNumber

data:

- char DocumentType[ST01]::LineItem.DocumentType
- char UnitOfMeasure[PO103]::LineItem.UnitOfMeasure
- char FedStockClass[4]::LineItem.FedStockClass
- char StdIndustrialClass[PO109]::LineItem.StdIndustrialClass
- char SRAN[SRAN\_LEN]::LineItem.SRANCode
- double Quantity::LineItem.Quantity
- char StatusCode[STATUS\_LEN]::LineItem.Status

}



### 3.2.1.47 LineItemStatus Object

```
Object LineItemStatus {
Relationships:
    Objects LineItem(StatusCode);
Exports:
    StatusCode
Members:
    key:
        char StatusCode[STATUS_LEN]::LineItemStatus.StatusCode
    data:
}
SimpleObject ListNumbers {
Relationships:
Exports:
Members:
    key:
    data:
        number FiscalYear::ListNumbers.FiscalYear
        long CurrentNumber::ListNumbers.CurrentNumber
}
```

### 3.2.1.48 Marks Object

```
Object Marks {
Relationships:
    IsA RelatedPaperwork(PaperworkID);
    Object MarksQualifier(Qualifier);
Exports:
    Qualifier
    PaperworkID
Members:
    key:
        number PaperworkID::Marks.PaperworkID
    data:
        char Qualifier[MAN01]::Marks.Qualifier
        char MarksAndNumbers[MAN02]::Marks.MarksAndNumbers
}
```

### 3.2.1.49 MarksQualifier Object

Object MarksQualifier {

Relationships:

Objects Marks(Qualifier);

Exports:

Qualifier

Members:

key:

char Qualifier[MAN01]::MarksQualifier.Qualifier

data:

}

### 3.2.1.50 MeasurementApplicationCode Object

Object MeasurementApplicationCode {

Relationships:

Objects MeasurementData(Application, ApplicationCode);

Exports:

Application

Members:

key:

char Application[MEA01]::MeasurementApplicationCode.Application

data:

}

### 3.2.1.51 MeasurementData Object

Object MeasurementData {

Relationships:

Object MeasurementApplicationCode(ApplicationCode, Application);

IsA RelatedPaperwork(PaperworkID);

Object UnitOfMeasure(UnitOfMeasure, UnitOfMeasureCode);

Exports:

ApplicationCode

PaperworkID

UnitOfMeasure

Members:

key:

number PaperworkID::MeasurementData.PaperworkID

data:

char ApplicationCode[MEA01]::MeasurementData.ApplicationCode

char TypeOfMeasurement[MEA02]::MeasurementData.TypeOfMeasurement

double MeasurementValue::MeasurementData.MeasurementValue

char UnitOfMeasure[MEA04]::MeasurementData.UnitOfMeasure

double MinimumValue::MeasurementData.MinimumValue

double MaximumValue::MeasurementData.MaximumValue

}

### 3.2.1.52 Message Object

```
Object Message {
Relationships:
    IsA Document(DocumentID);
    Objects MessageFrom(DocumentID);
Exports:
    DocumentID
Members:
    key:
        number DocumentID::Message.DocumentID
    data:
        char BuyerID[3]::Message.BuyerID
        dbDate MessageDate::Message.MessageDate
        char MessageNumber[30]::Message.MessageNumber
        char Subject[80]::Message.Subject
}
```

### 3.2.1.53 MessageFrom Object

```
Object MessageFrom {
Relationships:
    Object Message(DocumentID);
Exports:
    CommunicatorID
    DocumentID
Members:
    key:
        number DocumentID::MessageFrom.DocumentID
        long FromIndex::MessageFrom.FromIndex
    data:
        number SenderVendorID::MessageFrom.SenderVendorID
        char SenderLastName[N201]::MessageFrom.SenderLastName
        char SenderFirstName[N102]::MessageFrom.SenderFirstName
        char SenderAddress[N301]::MessageFrom.SenderAddress
        char SenderCity[N301]::MessageFrom.SenderCity
        char SenderState[N401]::MessageFrom.SenderState
        char SenderZIP[N403]::MessageFrom.SenderZIP
        char FirstContactID[CONTACTID]::MessageFrom.FirstContactID
        char SecondContactID[CONTACTID]::MessageFrom.SecondContactID
        char ThirdContactID[CONTACTID]::MessageFrom.ThirdContactID
}
```

### 3.2.1.54 MessageReference Object

```
Object MessageReference {  
  Relationships:  
  Exports:  
  Members:  
    key:  
      number DocumentID::MessageReference.DocumentID  
      char SolicitationNumber[7]::MessageReference.SolicitationNumber  
      char LineItem[4]::MessageReference.LineItem  
    data:  
  }
```



### 3.2.1.55 MessageTextBody Object

Object MessageTextBody {

Relationships:

Exports:

Members:

key:

number DocumentID::MessageTextBody.DocumentID

long BodyIndex::MessageTextBody.BodyIndex

data:

char TextBody[MAX\_TEXT]::MessageTextBody.TextBody

}

### 3.2.1.56 MessageTo Object

```
Object MessageTo {
Relationships:
Exports:
    TextID
    CommunicatorID
Members:
    key:
        number DocumentID::MessageTo.DocumentID
        long ToIndex::MessageTo.ToIndex
    data:
        number VendorID::MessageTo.VendorID
        char ReceiverLastName[N201]::MessageTo.ReceiverLastName
        char ReceiverFirstName[N102]::MessageTo.ReceiverFirstName
        char ReceiverAddress[N301]::MessageTo.ReceiverAddress
        char ReceiverCity[N301]::MessageTo.ReceiverCity
        char ReceiverState[N401]::MessageTo.ReceiverState
        char ReceiverZIP[N403]::MessageTo.ReceiverZIP
        char FirstContactID[CONTACTID]::MessageTo.FirstContactID
        char SecondContactID[CONTACTID]::MessageTo.SecondContactID
        char ThirdContactID[CONTACTID]::MessageTo.ThirdContactID
}
```

### **3.2.1.57 NegotiationAuthority**

Object NegotiationAuthority {

Relationships:

Objects BCASAward(Authority, NegotiationAuthority);

Exports:

Authority

Members:

key:

char Authority[NEGO\_AUTH]::NegotiationAuthority.Authority

data:

}

### 3.2.1.58 Nomenclature Object

Object Nomenclature {

Members:

key:

char StockNumber[15]::Nomenclature.StockNumber

data:

char Nomenclature07[40]::Nomenclature.Nomenclature07  
char Nomenclature08[40]::Nomenclature.Nomenclature08  
char Nomenclature09[40]::Nomenclature.Nomenclature09  
char Nomenclature10[40]::Nomenclature.Nomenclature10  
char Nomenclature11[40]::Nomenclature.Nomenclature11  
char Nomenclature12[40]::Nomenclature.Nomenclature12  
char Nomenclature13[40]::Nomenclature.Nomenclature13  
char Nomenclature14[40]::Nomenclature.Nomenclature14  
char Nomenclature15[40]::Nomenclature.Nomenclature15  
char Nomenclature16[40]::Nomenclature.Nomenclature16  
char Nomenclature17[40]::Nomenclature.Nomenclature17  
char Nomenclature18[40]::Nomenclature.Nomenclature18  
char Nomenclature19[40]::Nomenclature.Nomenclature19  
char Nomenclature20[40]::Nomenclature.Nomenclature20  
char Nomenclature21[40]::Nomenclature.Nomenclature21  
char Nomenclature22[40]::Nomenclature.Nomenclature22  
char Nomenclature23[40]::Nomenclature.Nomenclature23  
char Nomenclature24[40]::Nomenclature.Nomenclature24  
char Nomenclature25[40]::Nomenclature.Nomenclature25  
char Nomenclature26[40]::Nomenclature.Nomenclature26  
char Nomenclature27[40]::Nomenclature.Nomenclature27  
char Nomenclature28[40]::Nomenclature.Nomenclature28  
char Nomenclature29[40]::Nomenclature.Nomenclature29  
char Nomenclature30[40]::Nomenclature.Nomenclature30  
char Nomenclature31[40]::Nomenclature.Nomenclature31  
char Nomenclature32[40]::Nomenclature.Nomenclature32  
char Nomenclature33[40]::Nomenclature.Nomenclature33  
char Nomenclature34[40]::Nomenclature.Nomenclature34  
char Nomenclature35[40]::Nomenclature.Nomenclature35  
char Nomenclature36[40]::Nomenclature.Nomenclature36  
char Nomenclature37[40]::Nomenclature.Nomenclature37  
char Nomenclature38[40]::Nomenclature.Nomenclature38  
char Nomenclature39[40]::Nomenclature.Nomenclature39  
char Nomenclature40[40]::Nomenclature.Nomenclature40  
char Nomenclature41[40]::Nomenclature.Nomenclature41  
char Nomenclature42[40]::Nomenclature.Nomenclature42  
char Nomenclature43[40]::Nomenclature.Nomenclature43  
char Nomenclature44[40]::Nomenclature.Nomenclature44  
char Nomenclature45[40]::Nomenclature.Nomenclature45  
char Nomenclature46[40]::Nomenclature.Nomenclature46  
char Nomenclature47[40]::Nomenclature.Nomenclature47  
char Nomenclature48[40]::Nomenclature.Nomenclature48

}

### 3.2.1.59 Note Object

```
Object Note {
Relationships:
    Object NoteStatus(Status);
Exports:
    Status
Members:
    key:
        long NoteNumber::Note.NoteNumber
    data:
        char NoteText[MAX_TEXT]::Note.NoteText
        char IsElectronicMail[BOOLEAN_VALUE]::Note.IsElectronicMail
        char Status[NOTE_STATUS]::Note.Status
        dbDate CreationDate::Note.CreationDate
        number VendorID::Note.VendorID
        char BuyerID[BUYERID]::Note.BuyerID
        char EMailAddress[PER04]::Note.EMailAddress
}
```

### 3.2.1.60 NoteStatus Object

```
Object NoteStatus {
Relationships:
    Objects Note(Status);
Exports:
    Status
Members:
    key:
        char Status[NOTE_STATUS]::NoteStatus.Status
    data:
}
}
```

### 3.2.1.61 OpenPurchaseRequest Object

Object OpenPurchaseRequest {

Members:

key:

char RequisitionNumber[14]::OpenPurchaseRequest.RequisitionNumber  
char SolicitationNumber[7]::OpenPurchaseRequest.SolicitationNumber  
char LineItem[4]::OpenPurchaseRequest.LineItem

data:

char StockNumber[15]::OpenPurchaseRequest.StockNumber  
char SuspenseTime[3]::OpenPurchaseRequest.SuspenseTime  
char RequiredDeliveryDate[5]::OpenPurchaseRequest.RequiredDeliveryDate  
char DateReceived[5]::OpenPurchaseRequest.DateReceived  
char Priority[2]::OpenPurchaseRequest.Priority  
char Quantity[5]::OpenPurchaseRequest.Quantity  
char UnitOfIssue[2]::OpenPurchaseRequest.UnitOfIssue  
char RequisitionReturnIndicator[1]::OpenPurchaseRequest.RequisitionReturnIndicator  
char RequisitionReturnDate[5]::OpenPurchaseRequest.RequisitionReturnDate  
char DateCleared[5]::OpenPurchaseRequest.DateCleared  
char SignalCode[1]::OpenPurchaseRequest.SignalCode  
char SupplementalAddress[6]::OpenPurchaseRequest.SupplementalAddress  
char FundCode[2]::OpenPurchaseRequest.FundCode  
char RoutingID[3]::OpenPurchaseRequest.RoutingID  
char BuyerCode[3]::OpenPurchaseRequest.BuyerCode  
char EstimatedPrice[15]::OpenPurchaseRequest.EstimatedPrice  
char ProjectTitle[25]::OpenPurchaseRequest.ProjectTitle  
char AdviceCode[2]::OpenPurchaseRequest.AdviceCode  
char DemandCode[1]::OpenPurchaseRequest.DemandCode  
char SpwtInd[1]::OpenPurchaseRequest.SpwtInd  
char ControlDate[5]::OpenPurchaseRequest.ControlDate  
char ProjectCode[3]::OpenPurchaseRequest.ProjectCode

}

### 3.2.1.62 Opr Object

Object Opr {

Members:

key:

char RequisitionNumber[14]::Opr.RequisitionNumber  
char SolicitationNumber[7]::Opr.SolicitationNumber  
char LineItem[4]::Opr.LineItem

data:

char StockNumber[15]::Opr.StockNumber  
char SuspenseTime[3]::Opr.SuspenseTime  
char RequiredDeliveryDate[5]::Opr.RequiredDeliveryDate  
char DateReceived[5]::Opr.DateReceived  
char Priority[2]::Opr.Priority  
char Quantity[5]::Opr.Quantity  
char UnitOfIssue[2]::Opr.UnitOfIssue  
char RequisitionReturnIndicator[1]::Opr.RequisitionReturnIndicator  
char RequisitionReturnDate[5]::Opr.RequisitionReturnDate  
char DateCleared[5]::Opr.DateCleared  
char SignalCode[1]::Opr.SignalCode  
char SupplementalAddress[6]::Opr.SupplementalAddress  
char FundCode[2]::Opr.FundCode  
char RoutingID[3]::Opr.RoutingID  
char BuyerCode[3]::Opr.BuyerCode  
char EstimatedPrice[15]::Opr.EstimatedPrice  
char ProjectTitle[25]::Opr.ProjectTitle  
char AdviceCode[2]::Opr.AdviceCode  
char DemandCode[1]::Opr.DemandCode  
char SpwtInd[1]::Opr.SpwtInd  
char ControlDate[5]::Opr.ControlDate  
char ProjectCode[3]::Opr.ProjectCode

}



### 3.2.1.63 OrganizationalEntity Object

Object OrganizationalEntity {

Relationships:

Objects Award(EntityIDCode, BusEntityType);

Exports:

EntityIDCode

Members:

key:

char EntityIDCode[N101]::OrganizationalEntity.EntityIDCode

data:

char EntityDesc[N102]::OrganizationalEntity.EntityDescription

}

### 3.2.1.64 OriginalTransaction Object

Object OriginalTransaction {

Members:

key:

number DocumentID::OriginalTransaction.DocumentID

long OriginalTransactionID::OriginalTransaction.OriginalTransactionID

data:

char ApplicationAckCode[2]::OriginalTransaction.ApplicationAckCode

char ReferenceCode[2]::OriginalTransaction.ReferenceCode

char ReferenceNumber[30]::OriginalTransaction.ReferenceNumber

char ApplicationReceiverCode[15]::OriginalTransaction.ApplicationReceiverCode

char ApplicationSenderCode[15]::OriginalTransaction.ApplicationSenderCode

dbDate GroupDate::OriginalTransaction.GroupDate

dbDate GroupTime::OriginalTransaction.GroupTime

char GroupControlNumber[9]::OriginalTransaction.GroupControlNumber

char

TransactionSetControlNumber[9]::OriginalTransaction.TransactionSetControlNumber

char TransactionSetIdentifierCode[3]::OriginalTransaction.TransactionSetIdentifierCode

}

### 3.2.1.65 PTCType Object

```
Object PTCType {  
  Relationships:  
    Objects PolicyTermsAndConditions(PTCQual, PTCType);  
  Exports:  
    PTCQual  
  Members:  
    key:  
      char PTCQual[REF01]::PTCType.PTCQual  
    data:  
}  

```

### 3.2.1.66 Packaging Object

Object Packaging {

Relationships:

IsA RelatedPaperwork(PaperworkID);

Object PkgCharacteristicCode(PkgCharacteristicCode, PackagingCharCode);

Object PkgDescriptionCode(PkgDescriptionCode, PackagingDescCode);

Exports:

PkgDescriptionCode

PkgCharacteristicCode

PaperworkID

Members:

key:

number PaperworkID::Packaging.PaperworkID

data:

char PkgCharacteristicCode[PKG02]::Packaging.PkgCharacteristicCode

char PkgDescriptionCode[PKG04]::Packaging.PkgDescriptionCode

char PkgDescription[PKG05]::Packaging.PkgDescription

}

### 3.2.1.67 PaperworkType Object

Object PaperworkType {

Relationships:

Objects RelatedPaperwork(PaperworkType, PaperworkType);

Exports:

Type

Members:

key:

char PaperworkType[PAPERWORKTYPE]::PaperworkType.PaperworkType

data:

}

### 3.2.1.68 Part Object

```
Object Part {
Relationships:
    Object LineItem(DocumentID);
    Object LineItem(ItemNumber);
    Object PartIdentifier(PartIdentifier, PartType);
Exports:
    PartIdentifier
    ItemNumber
    DocumentID
Members:
    key:
        number DocumentID::Part.DocumentID
        char ItemNumber[PO101]::Part.ItemNumber
        char PartIdentifier[PO110]::Part.PartIdentifier
    data:
        char PartNumber[PO111]::Part.PartNumber
        char Manufacturer[MANUFACTURER]::Part.Manufacturer
        char ItemDescription[MAX_TEXT]::Part.ItemDescription
        char ServiceDescription[MAX_TEXT]::Part.ServiceDescription
}
```

### 3.2.1.69 PartIdentifier Object

```
Object PartIdentifier {
Relationships:
    Objects Part(PartType, PartIdentifier);
Exports:
    PartType
Members:
    key:
        char PartType[PO110]::PartIdentifier.PartType
    data:
        char PartDescription[MAX_TEXT]::PartIdentifier.PartDescription
}
```

### 3.2.1.70 Piins Object

```
Object Piins {  
  Relationships:  
  Exports:  
  Members:  
    key:      char PIIN[7]::Piins.PIIN  
    data:     char PiinStatus[6]::Piins.PiinStatus  
              char PiinType[6]::Piins.PiinType  
}
```



### 3.2.1.71 PkgCharacteristicCode Object

Object PkgCharacteristicCode {

Relationships:

Objects Packaging(PackagingCharCode, PkgCharacteristicCode);

Exports:

PackagingCharCode

Members:

key:

char PackagingCharCode[PKG02]::PkgCharacteristicCode.PackagingCharCode

data:

}

### 3.2.1.72 PkgDescriptionCode Object

Object PkgDescriptionCode {

Relationships:

Objects Packaging(PackagingDescCode, PkgDescriptionCode);

Exports:

PackagingDescCode

Members:

key:

char PackagingDescCode[PKG04]::PkgDescriptionCode.PackagingDescCode

data:

}

### 3.2.1.73 PolicyTermsAndConditions Object

Object PolicyTermsAndConditions {

Relationships:

Object PTCType(PTCType, PTCQual);

IsA RelatedPaperwork(PaperworkID);

Exports:

PaperworkID

PTCType

Members:

key:

number PaperworkID::PolicyTermsAndConditions.PaperworkID

data:

char PTCType[REF01]::PolicyTermsAndConditions.PTCType

char PTCRefNumber[REF02]::PolicyTermsAndConditions.PTCRefNumber

char PTCDescription[REF03]::PolicyTermsAndConditions.PTCDescription

}

### 3.2.1.74 PreferredAccess Object

Object PreferredAccess {

Relationships:

Objects Contact(CommNumQual, PreferredAccess);

Exports:

CommNumQual

Members:

key:

char CommNumQual[PER03]::PreferredAccess.CommNumQual

data:

}

### 3.2.1.75 PriorityGroup Object

```
Object PriorityGroup {  
  Relationships:  
  Exports:  
  Members:  
    key:      char PriorityID[PRIORITY_GROUP]::PriorityGroup.PriorityID  
    data:     long ReqResponseDays::PriorityGroup.ReqResponseDays  
              long ReqDeliveryDays::PriorityGroup.ReqDeliveryDays  
}
```

### 3.2.1.76 Project Object

```
Object Project {  
  Relationships:  
  Exports:  
    ProjectCode  
  Members:  
    key:      char ProjectCode[PROJECT_CODE]::Project.ProjectCode  
    data:  
}
```

### 3.2.1.77 PurchaseOrderAck Object

```
Object PurchaseOrderAck {  
  Relationships:  
    IsA Document(DocumentID);  
  Exports:  
    DocumentID  
  Members:  
    key:      number DocumentID::PurchaseOrderAck.DocumentID  
    data:  
}
```

### 3.2.1.78 PurchaseOrderChangeAckReq Object

Object PurchaseOrderChangeAckReq {

Relationships:

IsA Document(DocumentID);

Exports:

DocumentID

Members:

key:

number DocumentID::PurchaseOrderChangeAckReq.DocumentID

data:

}



### 3.2.1.79 Quote Object

Object Quote {

Relationships:

- Object Vendor(VendorID);
- Object Currency(CurrencyCode);
- IsA Document(DocumentID);
- Objects QuoteTerms(DocumentID);
- Object Contact(FirstContactID, ContactID);
- Object QuoteTypeCode(QuoteType);
- Object Vendor(QuoterCageCode, CageCode);
- Object Contact(SecondContactID, ContactID);
- Object Vendor(SellerCageCode, CageCode);
- Object Contact(ThirdContactID, ContactID);

Exports:

- QuoterCageCode
- SellerCageCode
- QuoteType
- ThirdContactID
- FirstContactID
- SecondContactID
- CurrencyCode
- DocumentID
- VendorID

Members:

key:

number DocumentID::Quote.DocumentID

data:

- number VendorID::Quote.VendorID
- char RFQRefNumber[BQR02]::Quote.RFQRefNumber
- char PriceQuoteRefNumber[REF03]::Quote.PriceQuoteRefNumber
- dbDate RFQEffectiveDate::Quote.RFQEffectiveDate
- dbDate QuoteEffectiveDate::Quote.QuoteEffectiveDate
- dbDate QuoteExpireDate::Quote.QuoteExpireDate
- char QuoteType[BQR06]::Quote.QuoteType
- char NotesAttached[BOOLEAN\_VALUE]::Quote.NotesAttached
- char CurrencyCode[CUR02]::Quote.CurrencyCode
- double ExchangeRate::Quote.ExchangeRate
- dbDate RateEffective::Quote.RateEffective
- dbDate RateExpires::Quote.RateExpires
- char ContractRefNumber[REF02]::Quote.ContractRefNumber
- char ContractDescription[REF03]::Quote.ContractDescription
- dbDate ContractExpireDate::Quote.ContractExpireDate
- char IsSmallBusiness[BOOLEAN\_VALUE]::Quote.IsSmallBusiness
- char FedSupplySchedNumber[REF02]::Quote.FedSupplySchedNumber
- dbDate FedSupplySchedDate::Quote.FedSupplySchedDate
- char SellerName[N102]::Quote.SellerName
- char SellerCageCode[N104]::Quote.SellerCageCode
- char SellerAddress[N301]::Quote.SellerAddress
- char SellerCity[N401]::Quote.SellerCity

char SellerState[N402]::Quote.SellerState  
char SellerZIPCode[N403]::Quote.SellerZIPCode  
char QuoterName[N102]::Quote.QuoterName  
char QuoterCageCode[N104]::Quote.QuoterCageCode  
char QuoterAddress[N301]::Quote.QuoterAddress  
char QuoterCity[N401]::Quote.QuoterCity  
char QuoterState[N402]::Quote.QuoterState  
char QuoterZIPCode[N403]::Quote.QuoterZIPCode  
char QuoterCountry[N404]::Quote.QuoterCountry  
char Electronic[BOOLEAN\_VALUE]::Quote.Electronic  
char FromFPI[BOOLEAN\_VALUE]::Quote.FromFPI  
char FromReqsContract[BOOLEAN\_VALUE]::Quote.FromReqsContract  
char FirstContactID[CONTACTID]::Quote.FirstContactID  
char SecondContactID[CONTACTID]::Quote.SecondContactID  
char ThirdContactID[CONTACTID]::Quote.ThirdContactID  
char QuoteDescription[MAX\_TEXT]::Quote.QuoteDescription

}

### 3.2.1.80 QuoteLineItem Object

Object QuoteLineItem {

Relationships:

- Object Document(DocumentID);
- Object FederalStockClass(FedStockClass, FedStockClassID);
- IsA LineItem(ItemNumber);
- Object UnitOfMeasure(UnitOfMeasure, UnitOfMeasureCode);
- Object UnitPriceCodeBasis(UnitPriceBasis);

Exports:

- FedStockClass
- ItemNumber
- UnitPriceBasis
- UnitOfMeasure

Members:

key:

- number DocumentID::QuoteLineItem.DocumentID
- char ItemNumber[PO101]::QuoteLineItem.ItemNumber

data:

- char

- IsFederalSupplySched[BOOLEAN\_VALUE]::QuoteLineItem.IsFederalSupplySched
- char ContractRefNumber[REF02]::QuoteLineItem.ContractRefNumber
- char ContractDescription[REF03]::QuoteLineItem.ContractDescription
- dbDate ContractExpireDate::QuoteLineItem.ContractExpireDate
- char ReferenceNumber[REF02]::QuoteLineItem.ReferenceNumber
- char ReferenceDescription[REF03]::QuoteLineItem.ReferenceDescription
- char ItemDescription01[80]::QuoteLineItem.ItemDescription01
- char ItemDescription02[80]::QuoteLineItem.ItemDescription02
- char ItemDescription03[80]::QuoteLineItem.ItemDescription03
- char ItemDescription04[80]::QuoteLineItem.ItemDescription04
- char ItemDescription05[80]::QuoteLineItem.ItemDescription05
- char ItemDescription06[80]::QuoteLineItem.ItemDescription06
- char ItemDescription07[80]::QuoteLineItem.ItemDescription07
- char ItemDescription08[80]::QuoteLineItem.ItemDescription08
- char ItemDescription09[80]::QuoteLineItem.ItemDescription09
- char ItemDescription10[80]::QuoteLineItem.ItemDescription10
- char ItemDescription11[80]::QuoteLineItem.ItemDescription11
- char ItemDescription12[80]::QuoteLineItem.ItemDescription12
- char ItemDescription13[80]::QuoteLineItem.ItemDescription13
- char ItemDescription14[80]::QuoteLineItem.ItemDescription14
- char ItemDescription15[80]::QuoteLineItem.ItemDescription15
- char ItemDescription16[80]::QuoteLineItem.ItemDescription16
- char ItemDescription17[80]::QuoteLineItem.ItemDescription17
- char ItemDescription18[80]::QuoteLineItem.ItemDescription18
- char ItemDescription19[80]::QuoteLineItem.ItemDescription19
- char ItemDescription20[80]::QuoteLineItem.ItemDescription20
- char ItemDescription21[80]::QuoteLineItem.ItemDescription21
- char ItemDescription22[80]::QuoteLineItem.ItemDescription22
- char ItemDescription23[80]::QuoteLineItem.ItemDescription23
- char ItemDescription24[80]::QuoteLineItem.ItemDescription24

```

char ItemDescription25[80]::QuoteLineItem.ItemDescription25
char ItemDescription26[80]::QuoteLineItem.ItemDescription26
char ItemDescription27[80]::QuoteLineItem.ItemDescription27
char ItemDescription28[80]::QuoteLineItem.ItemDescription28
char ItemDescription29[80]::QuoteLineItem.ItemDescription29
char ItemDescription30[80]::QuoteLineItem.ItemDescription30
char ItemDescription31[80]::QuoteLineItem.ItemDescription31
char ItemDescription32[80]::QuoteLineItem.ItemDescription32
char ItemDescription33[80]::QuoteLineItem.ItemDescription33
char ItemDescription34[80]::QuoteLineItem.ItemDescription34
char ItemDescription35[80]::QuoteLineItem.ItemDescription35
char ItemDescription36[80]::QuoteLineItem.ItemDescription36
char ItemDescription37[80]::QuoteLineItem.ItemDescription37
char ItemDescription38[80]::QuoteLineItem.ItemDescription38
char ItemDescription39[80]::QuoteLineItem.ItemDescription39
char ItemDescription40[80]::QuoteLineItem.ItemDescription40
char ItemDescription41[80]::QuoteLineItem.ItemDescription41
char ItemDescription42[80]::QuoteLineItem.ItemDescription42
char ItemDescription43[80]::QuoteLineItem.ItemDescription43
char ItemDescription44[80]::QuoteLineItem.ItemDescription44
char ItemDescription45[80]::QuoteLineItem.ItemDescription45
char ItemDescription46[80]::QuoteLineItem.ItemDescription46
char ItemDescription47[80]::QuoteLineItem.ItemDescription47
char ItemDescription48[80]::QuoteLineItem.ItemDescription48
dbDate DeliveryDate::QuoteLineItem.DeliveryDate
double Quantity::QuoteLineItem.Quantity
char UnitOfMeasure[PO103]::QuoteLineItem.UnitOfMeasure
double UnitPrice::QuoteLineItem.UnitPrice
char UnitPriceBasis[PO105]::QuoteLineItem.UnitPriceBasis
char FedStockClass[4]::QuoteLineItem.FedStockClass
char StdIndustrialClass[PO109]::QuoteLineItem.StdIndustrialClass
char PartListIncluded[BOOLEAN_VALUE]::QuoteLineItem.PartListIncluded
char VariationPercent[2]::QuoteLineItem.VariationPercent
char IsWinner[BOOLEAN_VALUE]::QuoteLineItem.IsWinner

```

```

}

```

### 3.2.1.81 QuoteTerms Object

Object QuoteTerms {

Relationships:

Object Quote(DocumentID);  
Object TermsMethods(PaymentMethod);  
Object TermsBasis(TermsBasis, BasisPeriod);

Exports:

DocumentID  
TermsBasis  
PaymentMethod

Members:

key:

number DocumentID::QuoteTerms.DocumentID

data:

char TermsBasis[ITD02]::QuoteTerms.TermsBasis  
double DiscountPercent::QuoteTerms.DiscountPercent  
dbDate DiscountDueDate::QuoteTerms.DiscountDueDate  
long DiscountDueDays::QuoteTerms.DiscountDueDays  
dbDate NetDueDate::QuoteTerms.NetDueDate  
long NetDueDays::QuoteTerms.NetDueDays  
long TotalDiscount::QuoteTerms.TotalDiscount  
dbDate DeferredDate::QuoteTerms.DeferredDate  
long DeferredAmount::QuoteTerms.DeferredAmount  
double InvoicePayablePercent::QuoteTerms.InvoicePayablePercent  
char Explanation[ITD12]::QuoteTerms.Explanation  
long DayOfMonth::QuoteTerms.DayOfMonth  
char PaymentMethod[ITD14]::QuoteTerms.PaymentMethod

}

### 3.2.1.82 QuoteTypeCode Object

Object QuoteTypeCode {

Relationships:

Objects Quote(QuoteType);

Exports:

QuoteType

Members:

key:

char QuoteType[BQR06]::QuoteTypeCode.QuoteType

data:

char TypeDescription[MAX\_TEXT]::QuoteTypeCode.TypeDescription

}

### 3.2.1.83 RedirectReason Object

Object RedirectReason {

Relationships:

Exports:

Reason

Members:

key:

char Reason[REASON\_CODE]::RedirectReason.Reason

data:

char Description[80]::RedirectReason.Description

}

### 3.2.1.84 RelatedPaperwork Object

Object RelatedPaperwork {

Relationships:

Object Document(DocumentID);  
Object LineItem(LineItemNumber, ItemNumber);  
Object PaperworkType(PaperworkType);

Exports:

LineItemNumber  
PaperworkID  
DocumentID  
PaperworkType

Members:

key:

number PaperworkID::RelatedPaperwork.PaperworkID

data:

number DocumentID::RelatedPaperwork.DocumentID  
char LineItemRelated[BOOLEAN\_VALUE]::RelatedPaperwork.LineItemRelated  
char LineItemNumber[4]::RelatedPaperwork.LineItemNumber  
char PaperworkType[PAPERWORKTYPE]::RelatedPaperwork.PaperworkType  
dbDate InclusionDate::RelatedPaperwork.InclusionDate

}



### 3.2.1.85 ReqForQuote Object

```
Object ReqForQuote {
Relationships:
    IsA Document(DocumentID);
Exports:
    DocumentID
Members:
    key:
        number DocumentID::ReqForQuote.DocumentID
    data:
        char DPASPriority[REF02]::ReqForQuote.DPASPriority
        char InternalOrderNumber[REF02]::ReqForQuote.InternalOrderNumber
        char SolicitationNumber[7]::ReqForQuote.SolicitationNumber
        char PurchaseReqNumber[REF02]::ReqForQuote.PurchaseReqNumber
        dbDate QuoteReceivedByDate::ReqForQuote.QuoteReceivedByDate
        dbTime QuoteReceivedByTime::ReqForQuote.QuoteReceivedByTime
        dbDate DeliveredByDate::ReqForQuote.DeliveredByDate
        char
SmallBusinessOrPurchase[BOOLEAN_VALUE]::ReqForQuote.SmallBusinessOrPurchase
        char SentToPublic[BOOLEAN_VALUE]::ReqForQuote.SentToPublic
        char RequestForQuoteDesc[NTE02]::ReqForQuote.RequestForQuoteDesc
        char Amended[BOOLEAN_VALUE]::ReqForQuote.Amended
}
```

### 3.2.1.86 ReqForQuoteLineItem Object

Object ReqForQuoteLineItem {

Relationships:

- Object Document(DocumentID);
- Object FederalStockClass(FedStockClass, FedStockClassID);
- IsA LineItem(ItemNumber);
- Object UnitOfMeasure(UnitOfMeasure, UnitOfMeasureCode);
- Object UnitPriceCodeBasis(UnitPriceBasis);

Exports:

- FedStockClass
- ItemNumber
- UnitPriceBasis
- UnitOfMeasure

Members:

key:

- number DocumentID::ReqForQuoteLineItem.DocumentID
- char ItemNumber[PO101]::ReqForQuoteLineItem.ItemNumber

data:

- char PurchaseReqNumber[REF02]::ReqForQuoteLineItem.PurchaseReqNumber
- char ItemDescription01[40]::ReqForQuoteLineItem.ItemDescription01
- char ItemDescription02[40]::ReqForQuoteLineItem.ItemDescription02
- char ItemDescription03[40]::ReqForQuoteLineItem.ItemDescription03
- char ItemDescription04[40]::ReqForQuoteLineItem.ItemDescription04
- char ItemDescription05[40]::ReqForQuoteLineItem.ItemDescription05
- char ItemDescription06[40]::ReqForQuoteLineItem.ItemDescription06
- char ItemDescription07[40]::ReqForQuoteLineItem.ItemDescription07
- char ItemDescription08[40]::ReqForQuoteLineItem.ItemDescription08
- char ItemDescription09[40]::ReqForQuoteLineItem.ItemDescription09
- char ItemDescription10[40]::ReqForQuoteLineItem.ItemDescription10
- char ItemDescription11[40]::ReqForQuoteLineItem.ItemDescription11
- char ItemDescription12[40]::ReqForQuoteLineItem.ItemDescription12
- char ItemDescription13[40]::ReqForQuoteLineItem.ItemDescription13
- char ItemDescription14[40]::ReqForQuoteLineItem.ItemDescription14
- char ItemDescription15[40]::ReqForQuoteLineItem.ItemDescription15
- char ItemDescription16[40]::ReqForQuoteLineItem.ItemDescription16
- char ItemDescription17[40]::ReqForQuoteLineItem.ItemDescription17
- char ItemDescription18[40]::ReqForQuoteLineItem.ItemDescription18
- char ItemDescription19[40]::ReqForQuoteLineItem.ItemDescription19
- char ItemDescription20[40]::ReqForQuoteLineItem.ItemDescription20
- char ItemDescription21[40]::ReqForQuoteLineItem.ItemDescription21
- char ItemDescription22[40]::ReqForQuoteLineItem.ItemDescription22
- char ItemDescription23[40]::ReqForQuoteLineItem.ItemDescription23
- char ItemDescription24[40]::ReqForQuoteLineItem.ItemDescription24
- char ItemDescription25[40]::ReqForQuoteLineItem.ItemDescription25
- char ItemDescription26[40]::ReqForQuoteLineItem.ItemDescription26
- char ItemDescription27[40]::ReqForQuoteLineItem.ItemDescription27
- char ItemDescription28[40]::ReqForQuoteLineItem.ItemDescription28
- char ItemDescription29[40]::ReqForQuoteLineItem.ItemDescription29
- char ItemDescription30[40]::ReqForQuoteLineItem.ItemDescription30

```

char ItemDescription31[40]::ReqForQuoteLineItem.ItemDescription31
char ItemDescription32[40]::ReqForQuoteLineItem.ItemDescription32
char ItemDescription33[40]::ReqForQuoteLineItem.ItemDescription33
char ItemDescription34[40]::ReqForQuoteLineItem.ItemDescription34
char ItemDescription35[40]::ReqForQuoteLineItem.ItemDescription35
char ItemDescription36[40]::ReqForQuoteLineItem.ItemDescription36
char ItemDescription37[40]::ReqForQuoteLineItem.ItemDescription37
char ItemDescription38[40]::ReqForQuoteLineItem.ItemDescription38
char ItemDescription39[40]::ReqForQuoteLineItem.ItemDescription39
char ItemDescription40[40]::ReqForQuoteLineItem.ItemDescription40
char ItemDescription41[40]::ReqForQuoteLineItem.ItemDescription41
char ItemDescription42[40]::ReqForQuoteLineItem.ItemDescription42
char ItemDescription43[40]::ReqForQuoteLineItem.ItemDescription43
char ItemDescription44[40]::ReqForQuoteLineItem.ItemDescription44
char ItemDescription45[40]::ReqForQuoteLineItem.ItemDescription45
char ItemDescription46[40]::ReqForQuoteLineItem.ItemDescription46
char ItemDescription47[40]::ReqForQuoteLineItem.ItemDescription47
char ItemDescription48[40]::ReqForQuoteLineItem.ItemDescription48
double Quantity::ReqForQuoteLineItem.Quantity
char UnitOfMeasure[PO103]::ReqForQuoteLineItem.UnitOfMeasure
double UnitPrice::ReqForQuoteLineItem.UnitPrice
char UnitPriceBasis[PO105]::ReqForQuoteLineItem.UnitPriceBasis
char FedStockClass[4]::ReqForQuoteLineItem.FedStockClass
char StdIndustrialClass[PO109]::ReqForQuoteLineItem.StdIndustrialClass
char PartListIncluded[BOOLEAN_VALUE]::ReqForQuoteLineItem.PartListIncluded
char FSCSuffix[3]::ReqForQuoteLineItem.FSCSuffix
char ShipToZIP[9]::ReqForQuoteLineItem.ShipToZIP
}

```

### 3.2.1.87 RequiredResponseTime Object

Object RequiredResponseTime {

Relationships:

Exports:

Members:

key:

double LowerDollarAmount::RequiredResponseTime.LowerDollarAmount

data:

long ReqResponseDays::RequiredResponseTime.ReqResponseDays

}

### 3.2.1.88 ReviewStatus Object

```
Object ReviewStatus {  
  Relationships:  
    Objects Document(Status, ReviewStatus);  
  Exports:  
    Status  
  Members:  
    key:  
      char Status[REVIEW_STATUS]::ReviewStatus.Status  
    data:  
}  

```

### 3.2.1.89 SADB U Object

```
Object SADB U {  
Relationships:  
Exports:  
Members:  
    key:  
        char LineItem[4]::SADB U.LineItem  
        number DocumentID::SADB U.DocumentID  
    data:  
        dbDate AwardDate::SADB U.AwardDate  
        char DissolutionReason[DISSO_REASON]::SADB U.DissolutionReason  
        double MoneySavedByDissolution::SADB U.MoneySavedByDissolution  
}
```

### 3.2.1.90 SendTo Object

```
Object SendTo {
Relationships:
    IsA RelatedPaperwork(PaperworkID);
Exports:
    PaperworkID
Members:
    key:
        number PaperworkID::SendTo.PaperworkID
    data:
        char ShipToCode[N101]::SendTo.ShipToCode
        char FirstName[N102]::SendTo.FirstName
        char LastName[N201]::SendTo.LastName
        char Address[N301]::SendTo.Address
        char City[N401]::SendTo.City
        char State[N402]::SendTo.State
        char ZIP[N403]::SendTo.ZIP
        char ShipToDescription[MAX_TEXT]::SendTo.ShipToDescription
}
```

### 3.2.1.91 Ship Object

Object Ship {

Relationships:

Members:

key:

char SRAN[SRAN\_LEN]::Ship.SRAN

data:

char DisbNum[6]::Ship.DisbNum

char OrgName[30]::Ship.OrgName

char OrgAddress[25]::Ship.OrgAddress

char OrgCity[25]::Ship.OrgCity

char OrgZip[9]::Ship.OrgZip

char PayOff[30]::Ship.PayOff

char PayAddress[30]::Ship.PayAddress

char PayCity[25]::Ship.PayCity

char PayZip[9]::Ship.PayZip

char AdminOff[30]::Ship.AdminOff

char AdminAddress[30]::Ship.AdminAddress

char AdminCity[25]::Ship.AdminCity

char AdminZip[9]::Ship.AdminZip

}



### 3.2.1.92 ShippingDeliveryTypes Object

Object ShippingDeliveryTypes {

Relationships:

Objects ShippingDocPackage(DocDeliveryMethods, DocDeliveryMethod);

Exports:

DocDeliveryMethods

Members:

key:

char DocDeliveryMethods[PWK02]::ShippingDeliveryTypes.DocDeliveryMethods

data:

char DeliveryDescription[MAX\_TEXT]::ShippingDeliveryTypes.DeliveryDescription

}

### 3.2.1.93 ShippingDocPackage Object

Object ShippingDocPackage {

Relationships:

Object ShippingDeliveryTypes(DocDeliveryMethod, DocDeliveryMethods);

Object ShippingDocTypes(DocumentType);

IsA RelatedPaperwork(PaperworkID);

Exports:

DocumentType

DocDeliveryMethod

PaperworkID

Members:

key:

number PaperworkID::ShippingDocPackage.PaperworkID

data:

char DocumentType[PWK01]::ShippingDocPackage.DocumentType

char DocDeliveryMethod[PWK02]::ShippingDocPackage.DocDeliveryMethod

long CopiesRequired::ShippingDocPackage.CopiesRequired

char WalshHealeyCompliant[2]::ShippingDocPackage.WalshHealeyCompliant

char AdditionalDesc[PWK07]::ShippingDocPackage.AdditionalDesc

}

### 3.2.1.94 ShippingDocTypes Object

Object ShippingDocTypes {

Relationships:

Objects ShippingDocPackage(DocumentType);

Exports:

DocumentType

Members:

key:

char DocumentType[PWK01]::ShippingDocTypes.DocumentType

data:

char DocumentDescription[MAX\_TEXT]::ShippingDocTypes.DocumentDescription

}

### 3.2.1.95 Signal Object

```
Object Signal {
Relationships:
Exports:
    SignalCode
Members:
    key:
        char SignalCode[SIGNAL_CODE]::Signal.SignalCode
    data:
}
SimpleObject SiteConfiguration {
Relationships:
Exports:
Members:
    key:
    data:
        char SiteAddress[PER04]::SiteConfiguration.SiteAddress
        char SiteName[PER02]::SiteConfiguration.SiteName
        char ReviewRequired[BOOLEAN_VALUE]::SiteConfiguration.ReviewRequired
        long AwardPostedPeriod::SiteConfiguration.AwardPostedPeriod
        long AwardAvailAfterShip::SiteConfiguration.AwardAvailAfterShip
        char
DeliveryDateCalculation[CALC_CODE]::SiteConfiguration.DeliveryDateCalculation
}
```

### 3.2.1.96 SolicitationHistory Object

Object SolicitationHistory {

Relationships:

- Object BCASPriority(BCASPriority, Priority);
- Object CancellationCode(CancellationCode, CancelCode);
- Object CompetitionCode(CompetitionCode, Competitive);
- Object Currency(Currency, BCASCurrency);

Exports:

- CompetitionCode
- SolicitationNumber
- Currency
- BCASPriority
- CancellationCode

Members:

key:

char StockNumber[15]::SolicitationHistory.StockNumber

data:

char SolicitationNumber[7]::SolicitationHistory.SolicitationNumber  
char PIIN[PIIN\_LEN]::SolicitationHistory.PIIN  
char SupplementalPIIN[PIIN\_SUPP]::SolicitationHistory.SupplementalPIIN  
char VendorCode[VENDOR\_CODE]::SolicitationHistory.VendorCode  
char CompetitionCode[COMP\_CODE]::SolicitationHistory.CompetitionCode  
dbDate AwardDate::SolicitationHistory.AwardDate  
char BCASPriority[BCAS\_PRIORITY]::SolicitationHistory.BCASPriority  
double Quantity::SolicitationHistory.Quantity  
char UnitOfIssue[UNIT\_OF\_ISSUE]::SolicitationHistory.UnitOfIssue  
double UnitPrice::SolicitationHistory.UnitPrice  
char Currency[CUR\_CODE]::SolicitationHistory.Currency  
dbDate EstimatedDeliveryDate::SolicitationHistory.EstimatedDeliveryDate  
char CancellationCode[CNX\_CODE]::SolicitationHistory.CancellationCode

}

### 3.2.1.97 SolicitationLineItem Object

Object SolicitationLineItem {

Members:

key:

char SolicitationNumber[7]::SolicitationLineItem.SolicitationNumber

char LineItem[4]::SolicitationLineItem.LineItem

data:

char FundCode[2]::SolicitationLineItem.FundCode

char ProjectTitle[25]::SolicitationLineItem.ProjectTitle

char ProjectCode[3]::SolicitationLineItem.ProjectCode

char SignalCode[1]::SolicitationLineItem.SignalCode

char SupplementalAddress[6]::SolicitationLineItem.SupplementalAddress

char BrandNameOrSoleSource[2]::SolicitationLineItem.BrandNameOrSoleSource

char Priority[2]::SolicitationLineItem.Priority

char ShipToSRAN[6]::SolicitationLineItem.ShipToSRAN

char BillToSRAN[6]::SolicitationLineItem.BillToSRAN

}

### 3.2.1.98 SolicitationLineItemError Object

```
Object SolicitationLineItemError {
Relationships:
    Objects Text(TextID, TextID);
Exports:
    TextID
Members:
    key:
        char SolicitationNumber[7]::SolicitationLineItemError.SolicitationNumber
        char LineItem[4]::SolicitationLineItemError.LineItem
    data:
        long TextID::SolicitationLineItemError.TextID
        char Subject[80]::SolicitationLineItemError.Subject
        dbDate ErrorDate::SolicitationLineItemError.ErrorDate
        char MessageRead[1]::SolicitationLineItemError.MessageRead
}
```

### 3.2.1.99 Stmtnt Object

Object Stmtnt {

Members:

key:

char StatementIndicator[2]::Stmtnt.StatementIndicator

data:

char StatementIndicator3[1]::Stmtnt.StatementIndicator3

char s1[65]::Stmtnt.s1

char s2[65]::Stmtnt.s2

char s3[65]::Stmtnt.s3

char s4[65]::Stmtnt.s4

char s5[65]::Stmtnt.s5

char s6[65]::Stmtnt.s6

char s7[65]::Stmtnt.s7

char s8[65]::Stmtnt.s8

char s9[65]::Stmtnt.s9

char s10[65]::Stmtnt.s10

char s11[65]::Stmtnt.s11

char s12[65]::Stmtnt.s12

char s13[65]::Stmtnt.s13

char s14[65]::Stmtnt.s14

char s15[65]::Stmtnt.s15

char s16[65]::Stmtnt.s16

char s17[65]::Stmtnt.s17

char s18[65]::Stmtnt.s18

char s19[65]::Stmtnt.s19

char s20[65]::Stmtnt.s20

char s21[65]::Stmtnt.s21

char s22[65]::Stmtnt.s22

char s23[65]::Stmtnt.s23

char s24[65]::Stmtnt.s24

char s25[65]::Stmtnt.s25

char s26[65]::Stmtnt.s26

char s27[65]::Stmtnt.s27

char s28[65]::Stmtnt.s28

char s29[65]::Stmtnt.s29

char s30[65]::Stmtnt.s30

}



### 3.2.1.100 TechnicalErrorDescription Object

Object TechnicalErrorDescription {

Members:

```
    key:      long OriginalTransactionID::TechnicalErrorDescription.OriginalTransactionID
    data:     char
ApplicationErrorCode[3]::TechnicalErrorDescription.ApplicationErrorCode
char ApplicationErrorMessage[60]::TechnicalErrorDescription.ApplicationErrorMessage
}
```

### 3.2.1.101 TermsBasis Object

Object TermsBasis {

Relationships:

Objects QuoteTerms(BasisPeriod, TermsBasis);

Exports:

BasisPeriod

Members:

key:

char TypeCode[ITD01]::TermsBasis.TypeCode

data:

char BasisPeriod[ITD02]::TermsBasis.BasisPeriod

char PeriodDescription[MAX\_TEXT]::TermsBasis.PeriodDescription

}

### 3.2.1.102 TermsMethods Object

```
Object TermsMethods {  
Relationships:  
    Objects QuoteTerms(PaymentMethod);  
Exports:  
    PaymentMethod  
Members:  
    key:  
        char PaymentMethod[ITD14]::TermsMethods.PaymentMethod  
    data:  
        char MethodDescription[MAX_TEXT]::TermsMethods.MethodDescription  
}
```

### 3.2.1.103 Text Object

```
Object Text {
Relationships:
    Object SolicitationLineItemError(TextID, TextID);
Exports:
    TextID
Members:
    key:
        long TextID::Text.TextID
        long LineIndex::Text.LineIndex
    data:
        char Body[255]::Text.Body
}
```

### 3.2.1.104 TransactionReference Object

Object TransactionReference {

Members:

```
    key:      char ReferenceCode[2]::TransactionReference.ReferenceCode
    data:     char ReferenceText[30]::TransactionReference.ReferenceText
}
```

### 3.2.1.105 TransactionSent Object

Object TransactionSent {

Members:

key:

number DocumentID::TransactionSent.DocumentID

data:

char InterchangeSenderID[15]::TransactionSent.InterchangeSenderID

char InterchangeReceiverID[15]::TransactionSent.InterchangeReceiverID

char

InterchangeReceiverIDQualifier[2]::TransactionSent.InterchangeReceiverIDQualifier

char ApplicationSenderID[15]::TransactionSent.ApplicationSenderID

char ApplicationReceiverID[15]::TransactionSent.ApplicationReceiverID

char SenderEmailAddress[255]::TransactionSent.SenderEmailAddress

char ReceiverEmailAddress[255]::TransactionSent.ReceiverEmailAddress

char InterchangeControlNumber[9]::TransactionSent.InterchangeControlNumber

char GroupControlNumber[9]::TransactionSent.GroupControlNumber

long TransactionSetControlNumber::TransactionSent.TransactionSetControlNumber

dbDate AdviceDate::TransactionSent.AdviceDate

dbDate AdviceTime::TransactionSent.AdviceTime

char PurposeCode[2]::TransactionSent.PurposeCode

}

### 3.2.1.106 TypeOfMeasurement Object

```
Object TypeOfMeasurement {  
  Relationships:  
    Objects MeasurementData(MeasurementType, TypeOfMeasurement);  
  Exports:  
    MeasurementType  
  Members:  
    key:  
      char MeasurementType[MEA02]::TypeOfMeasurement.MeasurementType  
    data:  
}  

```

### 3.2.1.107 Unit Object

Object Unit {

Members:

key:

char UnitOfIssue[2]::Unit.UnitOfIssue

char UnitOfMeasure[2]::Unit.UnitOfMeasure

data:

char UnitDescription[80]::Unit.UnitDescription

}



### 3.2.1.108 UnitOfMeasure Object

Object UnitOfMeasure {

Relationships:

- Objects AwardLineItem(UnitOfMeasureCode, UnitOfMeasure);
- Objects DeliverySchedule(UnitOfMeasureCode, UnitOfMeasure);
- Objects ItemDetails(UnitOfMeasureCode, LinearUnitOfMeasure);
- Objects ItemDetails(UnitOfMeasureCode, SizeUnitOfMeasure);
- Objects ItemDetails(UnitOfMeasureCode, VolumeUnitOfMeasure);
- Objects ItemDetails(UnitOfMeasureCode, WeightUnitOfMeasure);
- Objects MeasurementData(UnitOfMeasureCode, UnitOfMeasure);
- Objects QuoteLineItem(UnitOfMeasureCode, UnitOfMeasure);
- Objects ReqForQuoteLineItem(UnitOfMeasureCode, UnitOfMeasure);
- Objects Variations(UnitOfMeasureCode, UnitOfMeasure);

Exports:

UnitOfMeasureCode

Members:

key:

char UnitOfMeasureCode[MEA04]::UnitOfMeasure.UnitOfMeasureCode

data:

char UnitDescription[MAX\_TEXT]::UnitOfMeasure.UnitDescription

}

### 3.2.1.109 UnitPriceCodeBasis Object

Object UnitPriceCodeBasis {

Relationships:

Objects AwardLineItem(UnitPriceBasis);

Objects QuoteLineItem(UnitPriceBasis);

Objects ReqForQuoteLineItem(UnitPriceBasis);

Objects Variations(UnitPriceBasis, UnitPriceCodeBasis);

Exports:

UnitPriceType

Members:

key:

char UnitPriceBasis[PO105]::UnitPriceCodeBasis.UnitPriceBasis

data:

}

### 3.2.1.110 UserManagerDefaults Object

Object UserManagerDefaults {

Relationships:

Exports:

Members:

key:

char SiteAddress[26]::UserManagerDefaults.SiteAddress

data:

double EstimatedPriceLimit::UserManagerDefaults.EstimatedPriceLimit

double LargeBusinessPercentage::UserManagerDefaults.LargeBusinessPercentage

long OnlineDays::UserManagerDefaults.OnlineDays

char SendToPublic[BOOLEAN\_VALUE]::UserManagerDefaults.SendToPublic

long PurchaseOrderAckDays::UserManagerDefaults.PurchaseOrderAckDays

long AutoAckHours::UserManagerDefaults.AutoAckHours

long MaximumPriority::UserManagerDefaults.MaximumPriority

char NotificationAddress[255]::UserManagerDefaults.NotificationAddress

char Acknowledge840[BOOLEAN\_VALUE]::UserManagerDefaults.Acknowledge840

char Acknowledge850[BOOLEAN\_VALUE]::UserManagerDefaults.Acknowledge850

char Acknowledge864[BOOLEAN\_VALUE]::UserManagerDefaults.Acknowledge864

char UsersAllowed[BOOLEAN\_VALUE]::UserManagerDefaults.UsersAllowed

char MessageOfTheDay[255]::UserManagerDefaults.MessageOfTheDay

}

### 3.2.1.111 adrs Object

```
Object Vads {  
Relationships:  
Exports:  
Members:  
    key:  
        char Vendor[7]::Vads.Vendor  
    data:  
        char ConName[35]::Vads.ConName  
        char ConName2[35]::Vads.ConName2  
        char ConAdr1[35]::Vads.ConAdr1  
        char ConAdr2[30]::Vads.ConAdr2  
        char Zip[9]::Vads.Zip  
        char ExpDesg[1]::Vads.ExpDesg  
        char Contact[16]::Vads.Contact  
        char MinOrder[4]::Vads.MinOrder  
        char Phone[10]::Vads.Phone  
        char PhoneEx[4]::Vads.PhoneEx  
        char DunsNbr[9]::Vads.DunsNbr  
        char DunsNbr4[4]::Vads.DunsNbr4  
        char CustNbr[15]::Vads.CustNbr  
        char TaxId[9]::Vads.TaxId  
        char SolDate[5]::Vads.SolDate  
        char AwdDate[5]::Vads.AwdDate  
        char FaxNbr[10]::Vads.FaxNbr  
}
```

### 3.2.1.112 VariationType Object

Object VariationType {

Relationships:

Objects Variations(ChangeReason, VariationType);

Exports:

ChangeReason

Members:

key:

char ChangeReason[PO301]::VariationType.ChangeReason

data:

}

### 3.2.1.113 Variations Object

Object Variations {

Relationships:

- IsA RelatedPaperwork(PaperworkID);
- Object UnitOfMeasure(UnitOfMeasure, UnitOfMeasureCode);
- Object UnitPriceCodeBasis(UnitPriceCodeBasis, UnitPriceBasis);
- Object VariationType(VariationType, ChangeReason);

Exports:

- VariationType
- UnitPriceBasis
- PaperworkID
- UnitOfMeasure

Members:

key:

number PaperworkID::Variations.PaperworkID

data:

- char VariationType[PO301]::Variations.VariationType
- dbDate VariationDate::Variations.VariationDate
- double AlternatePrice::Variations.AlternatePrice
- char UnitPriceCodeBasis[PO305]::Variations.UnitPriceCodeBasis
- double Quantity::Variations.Quantity
- char UnitOfMeasure[PO307]::Variations.UnitOfMeasure
- char VariationDescription[PO308]::Variations.VariationDescription

}

### 3.2.1.114 Vendor Object

```
Object Vendor {
Relationships:
Exports:
    VendorID
Members:
    key:
        number VendorID::Vendor.VendorID
    data:
        char CageCode[17]::Vendor.CageCode
        char GovtPassword[30]::Vendor.GovtPassword
        char Temporary[BOOLEAN_VALUE]::Vendor.Temporary
        number PreviousVendorID::Vendor.PreviousVendorID
        char InterchangeReceiverQualifier[2]::Vendor.InterchangeReceiverQualifier
        char InterchangeReceiverID[15]::Vendor.InterchangeReceiverID
        char ApplicationReceiverID[15]::Vendor.ApplicationReceiverID
        char ElectronicMailAddress[255]::Vendor.ElectronicMailAddress
        char LocalSystemID[7]::Vendor.LocalSystemID
        date DateAssigned::Vendor.DateAssigned
        char ParentName[VENDOR_NAME]::Vendor.ParentName
        char IsAParent[BOOLEAN_VALUE]::Vendor.IsAParentCompany
        char IsAShelteredWorkshop[BOOLEAN_VALUE]::Vendor.IsAShelteredWorkshop
        char IsDebarred_Suspended[BOOLEAN_VALUE]::Vendor.IsDebarred_Suspended
        char PriStdIndustrialClass[PO109]::Vendor.PriStdIndustrialClass
        char OthStdIndustrialClass[PO109]::Vendor.OthStdIndustrialClass
        long NumberOfEmployees::Vendor.NumberOfEmployees
        char QuotesMadeAsSmallBus[BOOLEAN_VALUE]::Vendor.QuotesMadeAsSmallBus
        char SiteDesignation[17]::Vendor.SiteDesignation
}
```

### 3.2.1.115 VendorAward Object

```
Object VendorAward {  
Relationships:  
Exports:  
Members:  
    key:      number VendorID::VendorAward.VendorID  
    data:     char SiteID[SITENUMBER]::VendorAward.SiteID  
              number DocumentID::VendorAward.DocumentID  
              long ShippingDeliveryDelta::VendorAward.ShippingDeliveryDelta  
}
```



### 3.2.1.116 VendorContact Object

Object VendorContact {

Relationships:

Object Vendor(VendorID);

Object Contact(ContactID);

Exports:

ContactID

VendorID

Members:

key:

char ContactID[CONTACTID]::VendorContact.ContactID

data:

number VendorID::VendorContact.VendorID

long Priority::VendorContact.Priority

}

### 3.2.1.117 VendorHistory Object

Object VendorHistory {

Relationships:

Exports:

Members:

key:

number CurrentVendorID::VendorHistory.CurrentVendorID

data:

number OldVendorID::VendorHistory.OldVendorID

char OldName[VENDOR\_NAME]::VendorHistory.OldName

char OldCode[VENDOR\_CODE]::VendorHistory.OldCode

}

### 3.2.1.118 VendorQuoteLineItem Object

Object VendorQuoteLineItem {

Relationships:

Exports:

Members:

key:

number VendorID::VendorQuoteLineItem.VendorID

data:

char FedStockClass[4]::VendorQuoteLineItem.FedStockClass

}

---

### 3.2.2 The GATEC Database Schema

---

An FBI file is post-processed by a oraperl script that scans the definitions and parses the information into a database representation. The script will verify database names that are object names are being mapped to as well as type check the local object type to the database column type. A warning message will be generated for inconsistent types. The script is deficient in its ability to manage updates to the definitions. At present, the only update capability is to add new column mapping definitions. Updates to prior definitions are ignored. Therefore, the present method of managing updates to the database representation is to clean out the existing definitions and reload the FBI files. Because the FBI files are only intended as a portable source, the desirable goal is to initially load the definitions and then to manage all updates using database tools. A second oraperl utility is capable of generating new FBI files from the database representation if necessary. For complete details on the structure of the database representation, please refer to the *NORA Design Reference* manual.

---

### 3.2.3 Detailed Schema Description

---

The GATEC 2 database schema is described on the following pages.

### 3.2.3.1 ACCTG Table

Name	Null?	Type
-----	-----	-----
FUNDCODE		CHAR(2)
SRAN		CHAR(6)
ACCTGCLASS		CHAR(61)
ALLOT		CHAR(12)
EXPEND		CHAR(12)
MANAGEMENTNOTICE		CHAR(1)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.2 ACQUISITION Table

Name	Null?	Type
-----	-----	-----
RFQNUMBER	NOT NULL	CHAR(7)
SOLICITATIONNUMBER	NOT NULL	CHAR(7)
SITEID	NOT NULL	CHAR(6)
UTNNUMBER	NOT NULL	CHAR(16)
DPASPRIORITY		CHAR(30)
INTERNALORDERNUMBER		CHAR(30)
PURCHASEREQNUMBER		CHAR(30)
ASSIGNEDBUYER	NOT NULL	CHAR(3)
HOLDSTATUS		CHAR(2)
HOLDPERIOD		DATE
REVIEWSTATUS		CHAR(2)
PRIORITY		CHAR(2)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.3 ACTIVESTATUS Table

Name	Null?	Type
-----	-----	-----
STATUSIDENTIFIER	NOT NULL	NUMBER
STATUSDESCRIPTION		CHAR( 80 )

### 3.2.3.4 AWARD Table

Name	Null?	Type
-----	-----	-----
DOCUMENTID	NOT NULL	NUMBER
PURCHASETYPE		CHAR(2)
PURCHASEORDERNUMBER	NOT NULL	CHAR(22)
CALLDELIVERYORDERNUMBER		CHAR(30)
EFFECTIVEDATE		DATE
ACKNOWLEDGEMENT		CHAR(2)
AWARDDescription		CHAR(60)
BUYERCURRENCYCODE		CHAR(3)
BUYEREXCHANGERATE		NUMBER
BUYERRATEEFFECTIVE		DATE
BUYERRATEEXPIRES		DATE
SELLERCURRENCYCODE		CHAR(3)
SELLEREXCHANGERATE		NUMBER
SELLERRATEEFFECTIVE		DATE
SELLERRATEEXPIRES		DATE
INTERNALORDERNUMBER		CHAR(30)
PURCHASEREQNUMBER	NOT NULL	CHAR(30)
DPASPRIORITY		CHAR(30)
ACCTGNAPPRODATA		CHAR(30)
ACCTGCLASSREFNUMBER		CHAR(30)
QUOTEREFERENCENUMBER		CHAR(30)
QUOTEREFERENCEDATE		DATE
RFQREFERENCENUMBER	NOT NULL	CHAR(45)
RFQREFERENCEDATE		DATE
REQUIREDDELIVERYDATE		DATE
BUSENTITYTYPE		CHAR(2)
BUSENTITYNAME		CHAR(35)
BUSENTITYVENDORID		NUMBER
BUSENTITYDEPT		CHAR(35)
BUSENTITYADDRESS		CHAR(35)
BUSENTITYDEPT2		CHAR(35)
BUSENTITYADDRESS2		CHAR(35)
BUSENTITYCITY		CHAR(19)
BUSENTITYSTATE		CHAR(2)
BUSENTITYZIP		CHAR(9)
BIDNUMBER		CHAR(30)
BUYERSOFFICESYMBOL		CHAR(30)
CRITICALITYDESIGNATOR		CHAR(30)
FIRSTCONTACTID		CHAR(8)
SECONDCONTACTID		CHAR(8)
THIRDCONTACTID		CHAR(8)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)



### 3.2.3.5 AWARDLINEITEM Table

Name	Null?	Type
-----	-----	-----
DOCUMENTID	NOT NULL	NUMBER
ITEMNUMBER	NOT NULL	CHAR(11)
PRICETYPE		CHAR(2)
DPASPRIORITY		CHAR(30)
INTERNALORDERNUMBER		CHAR(30)
PURCHASEREQNUMBER		CHAR(30)
SINGLEDELIVERYDATE		NUMBER
DELIVERYDATE		DATE
TOTALLINEAMOUNT		NUMBER
QUANTITY		NUMBER
UNITOFMEASURE		CHAR(2)
UNITPRICE		NUMBER
UNITPRICEBASIS		CHAR(2)
FEDSTOCKCLASS	NOT NULL	CHAR(4)
STDINDUSTRIALCLASS		CHAR(30)
PARTLISTINCLUDED	NOT NULL	CHAR(1)
VARIATIONPERCENT		CHAR(2)
PURCHASEVARIATION		CHAR(1)
BUYERNAME		CHAR(35)
BUYERCAGECODE		CHAR(17)
BUYERDEPT		CHAR(35)
BUYERADDRESS		CHAR(35)
BUYERCITY		CHAR(25)
BUYERSTATE		CHAR(2)
BUYERZIP		CHAR(9)
SHIPTONAME		CHAR(35)
SHIPTOVENDORID		NUMBER
SHIPTODEPT		CHAR(35)
SHIPTOADDRESS		CHAR(35)
SHIPTOCITY		CHAR(25)
SHIPTOSTATE		CHAR(2)
SHIPTOZIP		CHAR(9)
BILLTONAME		CHAR(35)
BILLTOVENDORID		NUMBER
BILLTODEPT		CHAR(35)
BILLTOADDRESS		CHAR(35)
BILLTOCITY		CHAR(25)
BILLTOSTATE		CHAR(2)
BILLTOZIP		CHAR(9)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.6 AWARDPURCHASETYPE Table

Name	Null?	Type
-----	-----	-----
PURCHASETYPE	NOT NULL	CHAR ( 3 )
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR ( 8 )

### 3.2.3.7 BCASAWARD Table

Name	Null?	Type
-----	-----	-----
UTNNUMBER	NOT NULL	CHAR(16)
VENDORCODE		CHAR(7)
NEGOTIATIONAUTHORITY		CHAR(4)
COMPETITIONCODE		CHAR(17)
SOLICITATIONNUMBER		CHAR(7)
PIIN		CHAR(7)
ORDERSTATEMENTS		CHAR(30)
CONFIRMWITH		CHAR(15)
CONTRACTREFNUMBER		CHAR(30)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.8 BUYER Table

Name	Null?	Type
-----	-----	-----
LOCALSYSTEMID	NOT NULL	NUMBER
BUYERID	NOT NULL	CHAR(3)
LASTNAME	NOT NULL	CHAR(35)
FIRSTNAME	NOT NULL	CHAR(35)
MIDDLEINITIAL		CHAR(1)
PHONENUMBER		CHAR(25)
EMAILADDRESS		CHAR(25)
LEADSTATUS		CHAR(1)
DOWNLOAD		CHAR(1)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.9 CANCELLATIONCODE Table

Name	Null?	Type
-----	-----	-----
CANCELCODE	NOT NULL	CHAR ( 2 )
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR ( 8 )

### 3.2.3.10 CLAUSE Table

Name	Null?	Type
-----	-----	-----
PAPERWORKID	NOT NULL	NUMBER
CLAUSECERTIFICATION		CHAR(2)
CLAUSEREFNUMBER		CHAR(30)
CLAUSESOURCE		CHAR(45)
CLAUSEEXPLANATION		CHAR(45)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.11 COMMUNICATOR Table

Name	Null?	Type
-----	-----	-----
COMMUNICATORID	NOT NULL	NUMBER
LASTNAME	NOT NULL	CHAR(36)
FIRSTNAME		CHAR(36)
ADDITIONALNAME		CHAR(36)
ADDRESS		CHAR(36)
CITY		CHAR(20)
STATE		CHAR(3)
ZIP		CHAR(10)
FIRSTCONTACTID		CHAR(9)
SECONDCONTACTID		CHAR(9)
THIRDCONTACTID		CHAR(9)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.12 CONTACT Table

Name	Null?	Type
-----	-----	-----
CONTACTID	NOT NULL	CHAR(8)
NAME		CHAR(35)
PREFERREDACCESS		CHAR(2)
PHONENUMBER		CHAR(25)
FAXNUMBER		CHAR(25)
EMAILADDRESS		CHAR(25)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)



### 3.2.3.13 CONTROLSTANDARDS Table

Name	Null?	Type
-----	-----	-----
CONTROLSTANDARD	NOT NULL	CHAR(1)
DEFINITION		CHAR(255)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.14 CONTROLVERSION Table

Name	Null?	Type
-----	-----	-----
CONTROLVERSION	NOT NULL	CHAR(5)
DEFINITION		CHAR(255)
DMNUMBER		CHAR(8)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.15 DOCUMENT Table

Name	Null?	Type
-----	-----	-----
DOCUMENTID	NOT NULL	NUMBER
UTNNUMBER		CHAR(16)
TRANSACTIONNUMBER		CHAR(45)
DOCUMENTTYPE		CHAR(3)
DOCUMENTVERSION		CHAR(4)
X12REFERENCENUMBER		CHAR(45)
EFFECTIVEDATE		DATE
EXPIRATIONDATE		DATE
DOCUMENTSTATUS		CHAR(2)
REVIEWSTATUS		CHAR(2)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.16 DOCUMENTADDRESSEE Table

Name	Null?	Type
-----	-----	-----
DOCUMENTID	NOT NULL	NUMBER
VENDORID	NOT NULL	NUMBER
TRANSMITTALDATE		DATE
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR ( 8 )

### 3.2.3.17 DOCUMENTSENT Table

Name	Null?	Type
-----	-----	-----
DOCUMENTID	NOT NULL	NUMBER
GROUPCONTROLNUMBER	NOT NULL	NUMBER
TRANSACTIONSETCONTROLNUMBER		CHAR ( 9 )
INTERCHANGECONTROLNUMBER		CHAR ( 9 )
REJECTWARNINGSSENT		DATE
OVERDUEWARNINGSSENT		DATE
ISSUEDATE	NOT NULL	DATE
CHECKACKNOWLEDGEMENT		CHAR ( 1 )
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR ( 8 )

### 3.2.3.18 DOCUMENTSTATUS Table

Name	Null?	Type
-----	-----	-----
STATUS	NOT NULL	CHAR(17)
ACTIVE		CHAR(1)
DESCRIPTION		CHAR(255)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.19 DOCUMENTTYPE Table

Name	Null?	Type
TRANSACTIONSETID	NOT NULL	CHAR(3)
TYPEDESCRIPTION		CHAR(255)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.20 DOCUMENTVERSION Table

Name	Null?	Type
-----	-----	-----
VERSIONID	NOT NULL	CHAR(4)
VERSIONTYPE	NOT NULL	CHAR(17)
VERSIONDATE	NOT NULL	DATE
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)



### 3.2.3.21DOCUMENTVERSIONTYPE Table

Name	Null?	Type
-----	-----	-----
VERSIONTYPE	NOT NULL	CHAR(17)
VERSIONDESCRIPTION		CHAR(255)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.22 FREEONBOARD Table

Name	Null?	Type
-----	-----	-----
PAPERWORKID	NOT NULL	NUMBER
FOBTYPE		CHAR(2)
FOBDESCRIPTION		CHAR(80)
FOBACCEPTANCEPOINT		CHAR(2)
FOBALTERNATEINSPECTION		CHAR(1)
FOBINSPECTIONPOINT		CHAR(80)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.23 FSCSIC Table

Name	Null?	Type
-----	-----	-----
FEDSTOCKCLASS	NOT NULL	CHAR ( 4 )
STDINDUSTRIALCLASS	NOT NULL	CHAR ( 4 )
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR ( 8 )

### 3.2.3.24 FUNCTIONALGROUPHDR Table

Name	Null?	Type
-----	-----	-----
CAGECODE	NOT NULL	CHAR(17)
APPLICATIONSENDERSCODE		CHAR(15)
APPLICATIONRECEIVERSCODE		CHAR(15)
GROUPDATE		DATE
GROUPTIME		DATE
GROUPCONTROLNUMBER		CHAR(9)
RESPONSIBLEAGENCYCODE		CHAR(2)
VERSIONRELEASECODE		CHAR(12)

### 3.2.3.25 GSDEFAULTS Table

Name	Null?	Type
-----	-----	-----
DOCUMENTTYPE	NOT NULL	CHAR(4)
FUNCTIONALID	NOT NULL	CHAR(2)
APPLICATIONSENDER	NOT NULL	CHAR(15)
APPLICATIONRECEIVER	NOT NULL	CHAR(15)
GROUUPDATE	NOT NULL	DATE
GROUPTIME	NOT NULL	DATE
GROUPCONTROLNUMBER	NOT NULL	NUMBER
RESPONSIBLEAGENCY	NOT NULL	CHAR(2)
INTERCHANGEVERSION	NOT NULL	CHAR(12)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.26 HOLDSTATUS Table

Name	Null?	Type
-----	-----	-----
STATUS	NOT NULL	CHAR(2)
DESCRIPTION		CHAR(80)
DEFAULTDAYS		NUMBER
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.27 HOLIDAYS Table

Name	Null?	Type
-----	-----	-----
HOLIDAY	NOT NULL	DATE
DESCRIPTION		CHAR ( 255 )
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR ( 8 )

### 3.2.3.28 INTERCHANGECONTROLHDR Table

Name	Null?	Type
-----	-----	-----
CAGECODE	NOT NULL	CHAR(17)
SENDERID		CHAR(15)
CONTROLNUMBER		NUMBER
AUTHORIZATIONID		CHAR(2)
AUTHORIZATION		CHAR(10)
SECURITYID		CHAR(2)
SECURITY		CHAR(10)
INTERCHANGEID		CHAR(2)
RECEIVERID		CHAR(15)
INTERCHANGEDATE		DATE
INTERCHANGETIME		DATE
INTERCHANGECTLSTDS		CHAR(1)
INTERCHANGEVERSION		CHAR(5)
ACKREQUESTED		CHAR(1)
TESTINDICATOR		CHAR(1)
SUBELEMENTSEPARATOR		CHAR(1)
FUNCTIONALGROUPS		NUMBER
ACKCODE		CHAR(1)
NOTECODE		CHAR(3)
ELEMENTSEPARATOR		CHAR(1)



### 3.2.3.29 ISADEFAULTS Table

Name	Null?	Type
-----	-----	-----
DOCUMENTTYPE	NOT NULL	CHAR(4)
AUTHORIZATIONID	NOT NULL	CHAR(2)
AUTHORIZATION	NOT NULL	CHAR(10)
SECURITYID	NOT NULL	CHAR(2)
SECURITY	NOT NULL	CHAR(10)
SENDERIDQUALIFIER	NOT NULL	CHAR(2)
SENDERID	NOT NULL	CHAR(15)
RECEIVERIDQUALIFIER	NOT NULL	CHAR(2)
RECEIVERID	NOT NULL	CHAR(15)
INTERCHANGEDATE	NOT NULL	DATE
INTERCHANGETIME	NOT NULL	DATE
INTERCHANGECTLSTDS	NOT NULL	CHAR(1)
INTERCHANGEVERSION	NOT NULL	CHAR(5)
CONTROLNUMBER	NOT NULL	NUMBER
ACKREQUESTED	NOT NULL	CHAR(1)
TESTINDICATOR	NOT NULL	CHAR(1)
SUBELEMENTSEPARATOR	NOT NULL	CHAR(1)
ELEMENTSEPARATOR	NOT NULL	CHAR(1)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.30 ITEM Table

Name	Null?	Type
-----	-----	-----
STOCKNUMBER		CHAR(15)
SUPNOMENIND		CHAR(1)
CONTRIND		CHAR(1)
SUFFIX		CHAR(2)
UNITOFISSUE		CHAR(2)
BUYERCODE		CHAR(3)
CUSTOMERID		CHAR(1)
VARIATIONINQUANTITY		CHAR(2)
AUTOMATICPURCHASEORDER		CHAR(2)
BRANDNAMEORSOLESOURCE		CHAR(2)
RECDATE		CHAR(5)
COMMODITYASSIGNMENT		CHAR(1)
DATELASTAWARD		CHAR(5)
MANUFACTURERNAME		CHAR(30)
MANUFACTURERPART		CHAR(20)
NOMENCLATURE01		CHAR(40)
NOMENCLATURE02		CHAR(40)
NOMENCLATURE03		CHAR(40)
NOMENCLATURE04		CHAR(40)
NOMENCLATURE05		CHAR(40)
NOMENCLATURE06		CHAR(40)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.31 LINEITEM Table

Name	Null?	Type
-----	-----	-----
DOCUMENTID	NOT NULL	NUMBER
ITEMNUMBER	NOT NULL	CHAR(11)
DOCUMENTTYPE	NOT NULL	CHAR(3)
UNITOFMEASURE		CHAR(2)
FEDSTOCKCLASS		CHAR(4)
STDINDUSTRIALCLASS		CHAR(30)
SRANCODE		CHAR(6)
QUANTITY		NUMBER
STATUS		CHAR(17)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.32 MEASUREMENTAPPLICATIONCODE Table

Name	Null?	Type
-----	-----	-----
APPLICATION	NOT NULL	CHAR ( 3 )
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR ( 8 )

### 3.2.3.33 MEASUREMENTDATA Table

Name	Null?	Type
-----	-----	-----
PAPERWORKID	NOT NULL	NUMBER
APPLICATIONCODE		CHAR ( 2 )
TYPEOFMEASUREMENT		CHAR ( 3 )
MEASUREMENTVALUE		NUMBER
UNITOFMEASURE		CHAR ( 2 )
MINIMUMVALUE		NUMBER
MAXIMUMVALUE		NUMBER
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR ( 8 )

### 3.2.3.34 MESSAGE Table

Name	Null?	Type
-----	-----	-----
DOCUMENTID	NOT NULL	NUMBER
MESSAGEDATE		DATE
BUYERID		CHAR(3)
MESSAGENUMBER		CHAR(30)
SUBJECT		CHAR(80)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.35 MESSAGEFROM Table

Name	Null?	Type
-----	-----	----
DOCUMENTID	NOT NULL	NUMBER
FROMINDEX	NOT NULL	NUMBER
SENDERVENDORID		NUMBER
SENDERLASTNAME		CHAR(35)
SENDERFIRSTNAME		CHAR(35)
SENDERADDRESS		CHAR(35)
SENDERCITY		CHAR(19)
SENDERSTATE		CHAR(2)
SENDERZIP		CHAR(9)
FIRSTCONTACTID		CHAR(8)
SECONDCONTACTID		CHAR(8)
THIRDCONTACTID		CHAR(8)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.36 MESSAGEREFERENCE Table

Name	Null?	Type
-----	-----	-----
DOCUMENTID	NOT NULL	NUMBER
SOLICITATIONNUMBER		CHAR ( 7 )
LINEITEM		CHAR ( 4 )
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR ( 8 )



### 3.2.3.37 MESSAGETEXTBODY Table

Name	Null?	Type
-----	-----	-----
DOCUMENTID	NOT NULL	NUMBER
BODYINDEX	NOT NULL	NUMBER
TEXTBODY		CHAR ( 255 )
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR ( 8 )

### 3.2.3.38 MESSAGETO Table

Name	Null?	Type
-----	-----	----
DOCUMENTID	NOT NULL	NUMBER
TOINDEX	NOT NULL	NUMBER
VENDORID		NUMBER
RECEIVERLASTNAME		CHAR(35)
RECEIVERFIRSTNAME		CHAR(35)
RECEIVERADDRESS		CHAR(35)
RECEIVERCITY		CHAR(19)
RECEIVERSTATE		CHAR(2)
RECEIVERZIP		CHAR(9)
FIRSTCONTACTID		CHAR(8)
SECONDCONTACTID		CHAR(8)
THIRDCONTACTID		CHAR(8)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.39 NOMENCLATURE Table

Name	Null?	Type
-----	-----	-----
STOCKNUMBER	NOT NULL	CHAR(15)
NOMENCLATURE07		CHAR(40)
NOMENCLATURE08		CHAR(40)
NOMENCLATURE09		CHAR(40)
NOMENCLATURE10		CHAR(40)
NOMENCLATURE11		CHAR(40)
NOMENCLATURE12		CHAR(40)
NOMENCLATURE13		CHAR(40)
NOMENCLATURE14		CHAR(40)
NOMENCLATURE15		CHAR(40)
NOMENCLATURE16		CHAR(40)
NOMENCLATURE17		CHAR(40)
NOMENCLATURE18		CHAR(40)
NOMENCLATURE19		CHAR(40)
NOMENCLATURE20		CHAR(40)
NOMENCLATURE21		CHAR(40)
NOMENCLATURE22		CHAR(40)
NOMENCLATURE23		CHAR(40)
NOMENCLATURE24		CHAR(40)
NOMENCLATURE25		CHAR(40)
NOMENCLATURE26		CHAR(40)
NOMENCLATURE27		CHAR(40)
NOMENCLATURE28		CHAR(40)
NOMENCLATURE29		CHAR(40)
NOMENCLATURE30		CHAR(40)
NOMENCLATURE31		CHAR(40)
NOMENCLATURE32		CHAR(40)
NOMENCLATURE33		CHAR(40)
NOMENCLATURE34		CHAR(40)
NOMENCLATURE35		CHAR(40)
NOMENCLATURE36		CHAR(40)
NOMENCLATURE37		CHAR(40)
NOMENCLATURE38		CHAR(40)
NOMENCLATURE39		CHAR(40)
NOMENCLATURE40		CHAR(40)
NOMENCLATURE41		CHAR(40)
NOMENCLATURE42		CHAR(40)
NOMENCLATURE43		CHAR(40)
NOMENCLATURE44		CHAR(40)
NOMENCLATURE45		CHAR(40)
NOMENCLATURE46		CHAR(40)
NOMENCLATURE47		CHAR(40)
NOMENCLATURE48		CHAR(40)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.40 NOTE Table

Name	Null?	Type
-----	-----	-----
NOTENUMBER	NOT NULL	NUMBER
NOTETEXT		CHAR ( 255 )
ISELECTRONICMAIL		CHAR ( 1 )
STATUS		CHAR ( 2 )
CREATIONDATE		DATE
VENDORID		NUMBER
BUYERID		CHAR ( 3 )
EMAILADDRESS		CHAR ( 25 )
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR ( 8 )

### 3.2.3.41 OFFLINERFQS Table

Name	Null?	Type
-----	-----	-----
RFQNUMBER	NOT NULL	CHAR(7)
PIINNUMBER		CHAR(7)
ARCHIVEDATE	NOT NULL	DATE
RETRIEVEDATE		DATE
RETRIEVEACTIVEDAYS		NUMBER

### 3.2.3.42 OPR Table

Name	Null?	Type
-----	-----	-----
REQUISITIONNUMBER		CHAR(14)
STOCKNUMBER		CHAR(15)
SUSPENSETIME		CHAR(3)
REQUIREDDELIVERYDATE		CHAR(5)
DATERECEIVED		CHAR(5)
PRIORITY		CHAR(2)
QUANTITY		CHAR(5)
UNITOFISSUE		CHAR(2)
REQUISITIONRETURNINDICATOR		CHAR(1)
REQUISITIONRETURNDATE		CHAR(5)
DATECLEARED		CHAR(5)
SIGNALCODE		CHAR(1)
SUPPLEMENTALADDRESS		CHAR(6)
FUNDCODE		CHAR(2)
ROUTINGID		CHAR(3)
BUYERCODE		CHAR(3)
SOLICITATIONNUMBER		CHAR(7)
LINEITEM		CHAR(4)
ESTIMATEDPRICE		CHAR(15)
PROJECTTITLE		CHAR(25)
ADVICECODE		CHAR(2)
DEMANDCODE		CHAR(1)
SPWTIND		CHAR(1)
CONTROLDATE		CHAR(5)
PROJECTCODE		CHAR(3)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.43 ORIGINALTRANSACTION Table

Name	Null?	Type
-----	-----	-----
DOCUMENTID	NOT NULL	NUMBER
ORIGINALTRANSACTIONID	NOT NULL	NUMBER
APPLICATIONACKCODE	NOT NULL	CHAR(2)
REFERENCECODE		CHAR(2)
REFERENCENUMBER		CHAR(30)
APPLICATIONSENDERCODE		CHAR(15)
APPLICATIONRECEIVERCODE	NOT NULL	CHAR(15)
GROUUPDATE		DATE
GROUPTIME		DATE
GROUPCONTROLNUMBER		CHAR(9)
TRANSACTIONSETCONTROLNUMBER		CHAR(9)
TRANSACTIONSETIDENTIFIERCODE		CHAR(3)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.44 PACKAGING Table

Name	Null?	Type
-----	-----	-----
PAPERWORKID	NOT NULL	NUMBER
PKGCHARACTERISTICCODE		CHAR(5)
PKGDESCRIPTIONCODE		CHAR(7)
PKGDESCRIPTION		CHAR(80)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)



### 3.2.3.45 PART Table

Name	Null?	Type
-----	-----	-----
DOCUMENTID	NOT NULL	NUMBER
ITEMNUMBER	NOT NULL	CHAR(11)
PARTIDENTIFIER		CHAR(2)
PARTNUMBER		CHAR(30)
MANUFACTURER		CHAR(127)
ITEMDESCRIPTION		CHAR(255)
SERVICEDESCRIPTION		CHAR(255)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.46 PIINS Table

Name	Null?	Type
-----	-----	-----
PIIN	NOT NULL	CHAR ( 7 )
PIINSTATUS	NOT NULL	CHAR ( 6 )
PIINTYPE	NOT NULL	CHAR ( 6 )
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR ( 8 )

### 3.2.3.47 PREOPR Table

Name	Null?	Type
-----	-----	-----
REQUISITIONNUMBER		CHAR(14)
STOCKNUMBER		CHAR(15)
SUSPENSETIME		CHAR(3)
REQUIREDDELIVERYDATE		CHAR(5)
DATERECEIVED		CHAR(5)
PRIORITY		CHAR(2)
QUANTITY		CHAR(5)
UNITOFISSUE		CHAR(2)
REQUISITIONRETURNINDICATOR		CHAR(1)
REQUISITIONRETURNDATE		CHAR(5)
DATECLEARED		CHAR(5)
SIGNALCODE		CHAR(1)
SUPPLEMENTALADDRESS		CHAR(6)
FUNDCODE		CHAR(2)
ROUTINGID		CHAR(3)
BUYERCODE		CHAR(3)
SOLICITATIONNUMBER		CHAR(7)
LINEITEM		CHAR(4)
ESTIMATEDPRICE		CHAR(15)
PROJECTTITLE		CHAR(25)
ADVICECODE		CHAR(2)
DEMANDCODE		CHAR(1)
SPWTIND		CHAR(1)
CONTROLDATE		CHAR(5)
PROJECTCODE		CHAR(3)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.48 PRIORITYGROUP Table

Name	Null?	Type
-----	-----	-----
PRIORITYID	NOT NULL	CHAR( 30 )
REQRESPONSEDAYS		NUMBER
REQDELIVERYDAYS		NUMBER
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR( 8 )

### 3.2.3.49 QUOTE Table

Name	Null?	Type
DOCUMENTID	NOT NULL	NUMBER
VENDORID	NOT NULL	NUMBER
RFQREFNUMBER	NOT NULL	CHAR(45)
RFQEFFECTIVEDATE		DATE
QUOTEFFECTIVEDATE		DATE
QUOTEEXPIREDATE		DATE
QUOTETYPE		CHAR(2)
PRICEQUOTEREFNUMBER		CHAR(30)
NOTESATTACHED		CHAR(1)
CURRENCYCODE		CHAR(3)
EXCHANGERATE		NUMBER
RATEEFFECTIVE		DATE
RATEEXPIRES		DATE
CONTRACTREFNUMBER		CHAR(30)
CONTRACTDESCRIPTION		CHAR(80)
CONTRACTEXPIREDATE		DATE
ISSMALLBUSINESS	NOT NULL	CHAR(1)
FEDSUPPLYSCHEDNUMBER		CHAR(30)
FEDSUPPLYSCHEDDATE		DATE
SELLERNAME		CHAR(35)
SELLERCAGECODE		CHAR(17)
SELLERADDRESS		CHAR(35)
SELLERADDRESS2		CHAR(35)
SELLERCITY		CHAR(19)
SELLERSTATE		CHAR(2)
SELLERZIPCODE		CHAR(9)
SELLERCOUNTRY		CHAR(2)
QUOTERNAME		CHAR(35)
QUOTERCAGECODE		CHAR(17)
QUOTERADDRESS		CHAR(35)
QUOTERCITY		CHAR(19)
QUOTERSTATE		CHAR(2)
QUOTERZIPCODE		CHAR(9)
QUOTERCOUNTRY		CHAR(2)
ELECTRONIC		CHAR(1)
FROMFPI		CHAR(1)
FROMREQTS CONTRACT		CHAR(1)
FIRSTCONTACTID		CHAR(8)
SECONDCONTACTID		CHAR(8)
THIRDCONTACTID		CHAR(8)
QUOTEDESCRIPTION		CHAR(255)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.50 QUOTELINEITEM Table

Name	Null?	Type
-----	-----	-----
DOCUMENTID	NOT NULL	NUMBER
ITEMNUMBER	NOT NULL	CHAR(11)
ISFEDERALSUPPLYSCHED		CHAR(1)
CONTRACTREFNUMBER		CHAR(30)
CONTRACTDESCRIPTION		CHAR(80)
CONTRACTEXPIREDATE		DATE
REFERENCENUMBER		CHAR(30)
REFERENCEDESCRIPTION		CHAR(80)
ITEMDESCRIPTION01		CHAR(80)
ITEMDESCRIPTION02		CHAR(80)
ITEMDESCRIPTION03		CHAR(80)
ITEMDESCRIPTION04		CHAR(80)
ITEMDESCRIPTION05		CHAR(80)
ITEMDESCRIPTION06		CHAR(80)
ITEMDESCRIPTION07		CHAR(80)
ITEMDESCRIPTION08		CHAR(80)
ITEMDESCRIPTION09		CHAR(80)
ITEMDESCRIPTION10		CHAR(80)
ITEMDESCRIPTION11		CHAR(80)
ITEMDESCRIPTION12		CHAR(80)
ITEMDESCRIPTION13		CHAR(80)
ITEMDESCRIPTION14		CHAR(80)
ITEMDESCRIPTION15		CHAR(80)
ITEMDESCRIPTION16		CHAR(80)
ITEMDESCRIPTION17		CHAR(80)
ITEMDESCRIPTION18		CHAR(80)
ITEMDESCRIPTION19		CHAR(80)
ITEMDESCRIPTION20		CHAR(80)
ITEMDESCRIPTION21		CHAR(80)
ITEMDESCRIPTION22		CHAR(80)
ITEMDESCRIPTION23		CHAR(80)
ITEMDESCRIPTION24		CHAR(80)
ITEMDESCRIPTION25		CHAR(80)
ITEMDESCRIPTION26		CHAR(80)
ITEMDESCRIPTION27		CHAR(80)
ITEMDESCRIPTION28		CHAR(80)
ITEMDESCRIPTION29		CHAR(80)
ITEMDESCRIPTION30		CHAR(80)
ITEMDESCRIPTION31		CHAR(80)
ITEMDESCRIPTION32		CHAR(80)
ITEMDESCRIPTION33		CHAR(80)
ITEMDESCRIPTION34		CHAR(80)
ITEMDESCRIPTION35		CHAR(80)
ITEMDESCRIPTION36		CHAR(80)
ITEMDESCRIPTION37		CHAR(80)
ITEMDESCRIPTION38		CHAR(80)
ITEMDESCRIPTION39		CHAR(80)
ITEMDESCRIPTION40		CHAR(80)
ITEMDESCRIPTION41		CHAR(80)
ITEMDESCRIPTION42		CHAR(80)

ITEMDESCRIPTION43		CHAR(80)
ITEMDESCRIPTION44		CHAR(80)
ITEMDESCRIPTION45		CHAR(80)
ITEMDESCRIPTION46		CHAR(80)
ITEMDESCRIPTION47		CHAR(80)
ITEMDESCRIPTION48		CHAR(80)
DELIVERYDATE		DATE
QUANTITY	NOT NULL	NUMBER
UNITOFMEASURE	NOT NULL	CHAR(2)
UNITPRICE		NUMBER
UNITPRICEBASIS		CHAR(2)
FEDSTOCKCLASS		CHAR(4)
STDINDUSTRIALCLASS		CHAR(30)
PARTLISTINCLUDED		CHAR(1)
VARIATIONPERCENT		CHAR(2)
ISWINNER		CHAR(1)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.51 QUOTETERMS Table

Name	Null?	Type
-----	-----	-----
DOCUMENTID	NOT NULL	NUMBER
TERMSBASIS		CHAR ( 2 )
DISCOUNTPERCENT		NUMBER
DISCOUNTDUEDATE		DATE
DISCOUNTDUEHOURS		NUMBER
NETDUEHOURS		DATE
NETDUEHOURS		NUMBER
TOTALDISCOUNT		NUMBER
DEFERREDDATE		DATE
DEFERREDAMOUNT		NUMBER
INVOICEPAYABLEPERCENT		NUMBER
EXPLANATION		CHAR ( 80 )
DAYOFMONTH		NUMBER
PAYMENTMETHOD		CHAR ( 1 )
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR ( 8 )



### 3.2.3.52 RELATEDPAPERWORK Table

Name	Null?	Type
-----	-----	-----
DOCUMENTID	NOT NULL	NUMBER
PAPERWORKID	NOT NULL	NUMBER
PAPERWORKTYPE	NOT NULL	CHAR ( 2 )
LINEITEMRELATED		CHAR ( 1 )
LINEITEMNUMBER		CHAR ( 4 )
INCLUSIONDATE		DATE
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR ( 8 )

### 3.2.3.53 REQFORQUOTE Table

Name	Null?	Type
-----	-----	-----
DOCUMENTID	NOT NULL	NUMBER
DPASPRIORITY		CHAR(30)
INTERNALORDERNUMBER		CHAR(30)
SOLICITATIONNUMBER	NOT NULL	CHAR(7)
PURCHASEREQNUMBER		CHAR(30)
QUOTERECEIVEDBYDATE		DATE
QUOTERECEIVEDBYTIME		DATE
DELIVEREDBYDATE		DATE
SMALLBUSINESSORPURCHASE		CHAR(1)
SENTTOPUBLIC		CHAR(1)
REQUESTFORQUOTEDESC		CHAR(60)
AMENDED	NOT NULL	CHAR(1)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.54 REQFORQUOTELINEITEM Table

Name	Null?	Type
-----	-----	-----
DOCUMENTID	NOT NULL	NUMBER
ITEMNUMBER	NOT NULL	CHAR(11)
PURCHASEREQNUMBER	NOT NULL	CHAR(30)
ITEMDESCRIPTION01		CHAR(40)
ITEMDESCRIPTION02		CHAR(40)
ITEMDESCRIPTION03		CHAR(40)
ITEMDESCRIPTION04		CHAR(40)
ITEMDESCRIPTION05		CHAR(40)
ITEMDESCRIPTION06		CHAR(40)
ITEMDESCRIPTION07		CHAR(40)
ITEMDESCRIPTION08		CHAR(40)
ITEMDESCRIPTION09		CHAR(40)
ITEMDESCRIPTION10		CHAR(40)
ITEMDESCRIPTION11		CHAR(40)
ITEMDESCRIPTION12		CHAR(40)
ITEMDESCRIPTION13		CHAR(40)
ITEMDESCRIPTION14		CHAR(40)
ITEMDESCRIPTION15		CHAR(40)
ITEMDESCRIPTION16		CHAR(40)
ITEMDESCRIPTION17		CHAR(40)
ITEMDESCRIPTION18		CHAR(40)
ITEMDESCRIPTION19		CHAR(40)
ITEMDESCRIPTION20		CHAR(40)
ITEMDESCRIPTION21		CHAR(40)
ITEMDESCRIPTION22		CHAR(40)
ITEMDESCRIPTION23		CHAR(40)
ITEMDESCRIPTION24		CHAR(40)
ITEMDESCRIPTION25		CHAR(40)
ITEMDESCRIPTION26		CHAR(40)
ITEMDESCRIPTION27		CHAR(40)
ITEMDESCRIPTION28		CHAR(40)
ITEMDESCRIPTION29		CHAR(40)
ITEMDESCRIPTION30		CHAR(40)
ITEMDESCRIPTION31		CHAR(40)
ITEMDESCRIPTION32		CHAR(40)
ITEMDESCRIPTION33		CHAR(40)
ITEMDESCRIPTION34		CHAR(40)
ITEMDESCRIPTION35		CHAR(40)
ITEMDESCRIPTION36		CHAR(40)
ITEMDESCRIPTION37		CHAR(40)
ITEMDESCRIPTION38		CHAR(40)
ITEMDESCRIPTION39		CHAR(40)
ITEMDESCRIPTION40		CHAR(40)
ITEMDESCRIPTION41		CHAR(40)
ITEMDESCRIPTION42		CHAR(40)
ITEMDESCRIPTION43		CHAR(40)
ITEMDESCRIPTION44		CHAR(40)
ITEMDESCRIPTION45		CHAR(40)
ITEMDESCRIPTION46		CHAR(40)
ITEMDESCRIPTION47		CHAR(40)

ITEMDESCRIPTION48		CHAR(40)
QUANTITY	NOT NULL	NUMBER
UNITOFMEASURE		CHAR(2)
UNITPRICE		NUMBER
UNITPRICEBASIS		CHAR(2)
FEDSTOCKCLASS	NOT NULL	CHAR(4)
STDINDUSTRIALCLASS		CHAR(30)
PARTLISTINCLUDED		CHAR(1)
FSCSUFFIX		CHAR(2)
SHIPTOZIP		CHAR(9)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.55 REVIEWSTATUS Table

Name	Null?	Type
-----	-----	-----
STATUS	NOT NULL	CHAR(17)
DESCRIPTION		CHAR(255)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.56 SADB Table

Name	Null?	Type
-----	-----	-----
LINEITEM	NOT NULL	CHAR ( 4 )
DOCUMENTID	NOT NULL	NUMBER
AWARDDATE		DATE
DISSOLUTIONREASON		CHAR ( 2 )
MONEYSAVEDBYDISSOLUTION		NUMBER
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR ( 8 )

### 3.2.3.57 SHIP Table

Name	Null?	Type
-----	-----	-----
SRAN		CHAR ( 6 )
DISBNUM		CHAR ( 6 )
ORGNAME		CHAR ( 30 )
ORGADDRESS		CHAR ( 30 )
ORGCITY		CHAR ( 25 )
ORGZIP		CHAR ( 9 )
PAYOFF		CHAR ( 30 )
PAYADDRESS		CHAR ( 30 )
PAYCITY		CHAR ( 25 )
PAYZIP		CHAR ( 9 )
ADMINOFF		CHAR ( 30 )
ADMINADDRESS		CHAR ( 30 )
ADMINCITY		CHAR ( 25 )
ADMINZIP		CHAR ( 9 )
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR ( 8 )

### 3.2.3.58 SHIPPINGDOCPACKAGE Table

Name	Null?	Type
-----	-----	-----
PAPERWORKID	NOT NULL	NUMBER
DOCUMENTTYPE	NOT NULL	CHAR( 2 )
DOCDELIVERYMETHOD		CHAR( 2 )
COPIESREQUIRED		NUMBER
WALSHHEALEYCOMPLIANT		CHAR( 2 )
ADDITIONALDESC		CHAR( 80 )
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR( 8 )



### 3.2.3.59 ITECONFIGURATION Table

Name	Null?	Type
-----	-----	-----
SITEADDRESS	NOT NULL	CHAR( 25 )
SITENAME	NOT NULL	CHAR( 35 )
REVIEWREQUIRED	NOT NULL	CHAR( 1 )
AWARDPOSTEDPERIOD	NOT NULL	NUMBER
AWARDAVAILAFTERSHIP	NOT NULL	NUMBER
DELIVERYDATECALCULATION	NOT NULL	CHAR( 1 )
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR( 8 )

### 3.2.3.60 SOLICITATIONHISTORY Table

Name	Null?	Type
-----	-----	-----
STOCKNUMBER	NOT NULL	CHAR(15)
SOLICITATIONNUMBER	NOT NULL	CHAR(7)
PIIN		CHAR(7)
SUPPLEMENTALPIIN		CHAR(4)
VENDORCODE		CHAR(7)
COMPETITIONCODE		CHAR(1)
AWARDDATE		DATE
BCASPRIORITY		CHAR(2)
QUANTITY		NUMBER
UNITOFISSUE		CHAR(2)
UNITPRICE		NUMBER
CURRENCY		CHAR(17)
ESTIMATEDDELIVERYDATE		DATE
CANCELLATIONCODE		CHAR(1)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.61 SOLICITATIONLINEITEM Table

Name	Null?	Type
-----	-----	-----
SOLICITATIONNUMBER		CHAR ( 7 )
LINEITEM		CHAR ( 4 )
FUNDCODE		CHAR ( 2 )
PROJECTTITLE		CHAR ( 25 )
PROJECTCODE		CHAR ( 3 )
SIGNALCODE		CHAR ( 1 )
SUPPLEMENTALADDRESS		CHAR ( 6 )
BRANDNAMEORSOLESOURCE		CHAR ( 2 )
PRIORITY		CHAR ( 2 )
SHIPTOSRAN		CHAR ( 6 )
BILLTOSRAN		CHAR ( 6 )
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR ( 8 )

### 3.2.3.62 SOLICITATIONLINEITEMERROR Table

Name	Null?	Type
-----	-----	-----
SOLICITATIONNUMBER		CHAR ( 7 )
LINEITEM		CHAR ( 4 )
TEXTID	NOT NULL	NUMBER
SUBJECT		CHAR ( 80 )
ERRORDATE	NOT NULL	DATE
MESSAGEREAD		CHAR ( 1 )
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR ( 8 )

### 3.2.3.63 STATUSOPERATION Table

Name	Null?	Type
-----	-----	-----
STATUSIDENTIFIER	NOT NULL	NUMBER
STATUSDESCRIPTION		CHAR ( 80 )

### 3.2.3.64 STMNT Table

Name	Null?	Type
STATEMENTINDICATOR		CHAR(2)
STATEMENTINDICATOR3		CHAR(1)
S1		CHAR(65)
S2		CHAR(65)
S3		CHAR(65)
S4		CHAR(65)
S5		CHAR(65)
S6		CHAR(65)
S7		CHAR(65)
S8		CHAR(65)
S9		CHAR(65)
S10		CHAR(65)
S11		CHAR(65)
S12		CHAR(65)
S13		CHAR(65)
S14		CHAR(65)
S15		CHAR(65)
S16		CHAR(65)
S17		CHAR(65)
S18		CHAR(65)
S19		CHAR(65)
S20		CHAR(65)
S21		CHAR(65)
S22		CHAR(65)
S23		CHAR(65)
S24		CHAR(65)
S25		CHAR(65)
S26		CHAR(65)
S27		CHAR(65)
S28		CHAR(65)
S29		CHAR(65)
S30		CHAR(65)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.65 TECHNICALERRORDESCRIPTION Table

Name	Null?	Type
-----	-----	-----
ORIGINALTRANSACTIONID	NOT NULL	NUMBER
APPLICATIONERRORCONDITIONCODE	NOT NULL	CHAR( 3 )
APPLICATIONERRORMESSAGE	NOT NULL	CHAR( 60 )
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR( 8 )

### 3.2.3.66 TERMSBASIS Table

Name	Null?	Type
-----	-----	-----
TYPECODE	NOT NULL	CHAR(2)
BASISPERIOD		CHAR(2)
PERIODDESCRIPTION		CHAR(255)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)



### 3.2.3.67 TEXT Table

Name	Null?	Type
-----	-----	-----
TEXTID	NOT NULL	NUMBER
LINEINDEX	NOT NULL	NUMBER
BODY		CHAR ( 255 )
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR ( 8 )

### 3.2.3.68 TRANSACTIONSENT Table

Name	Null?	Type
-----	-----	-----
DOCUMENTID	NOT NULL	NUMBER
INTERCHANGESENDERID	NOT NULL	CHAR(15)
INTERCHANGERECEIVERID	NOT NULL	CHAR(15)
INTERCHANGERECEIVERIDQUALIFIER	NOT NULL	CHAR(2)
APPLICATIONSENDERID	NOT NULL	CHAR(15)
APPLICATIONRECEIVERID	NOT NULL	CHAR(15)
SENDEREMAILADDRESS	NOT NULL	CHAR(255)
RECEIVEREMAILADDRESS	NOT NULL	CHAR(255)
INTERCHANGECONTROLNUMBER	NOT NULL	CHAR(9)
GROUPCONTROLNUMBER	NOT NULL	CHAR(9)
TRANSACTIONSETCONTROLNUMBER	NOT NULL	NUMBER
ADVISEDATE	NOT NULL	DATE
ADVICTIME	NOT NULL	DATE
PURPOSECODE	NOT NULL	CHAR(2)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.69 UNIT Table

Name	Null?	Type
-----	-----	-----
UNITOFISSUE	NOT NULL	CHAR(2)
UNITOFMEASURE	NOT NULL	CHAR(2)
UNITDESCRIPTION	NOT NULL	CHAR(80)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.70 UNITOFMEASURE Table

Name	Null?	Type
-----	-----	-----
UNITOFMEASURECODE	NOT NULL	CHAR ( 2 )
UNITDESCRIPTION		CHAR ( 255 )
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR ( 8 )

### 3.2.3.71 USERMANAGERDEFAULTS Table

Name	Null?	Type
-----	-----	-----
SITEADDRESS	NOT NULL	CHAR(26)
ESTIMATEDPRICELIMIT	NOT NULL	NUMBER
LARGEBUSINESSPERCENTAGE		NUMBER
ONLINEDAYS		NUMBER
SENDTOPUBLIC		CHAR(1)
PURCHASEORDERACKDAYS		NUMBER
AUTOACKHOURS	NOT NULL	NUMBER
MAXIMUMPRIORITY	NOT NULL	NUMBER
NOTIFICATIONADDRESS		CHAR(255)
ACKNOWLEDGE840		CHAR(1)
ACKNOWLEDGE850		CHAR(1)
ACKNOWLEDGE864		CHAR(1)
USERSALLOWED	NOT NULL	CHAR(1)
MESSAGEOFTHEDAY		CHAR(255)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.72 VADRS Table

Name	Null?	Type
-----	-----	-----
VENDOR		CHAR(7)
CONNAME		CHAR(35)
CONNAME2		CHAR(35)
CONADR1		CHAR(35)
CONADR2		CHAR(30)
ZIP		CHAR(9)
EXPDESG		CHAR(1)
CONTACT		CHAR(16)
MINORDER		CHAR(4)
PHONE		CHAR(10)
PHONEEX		CHAR(4)
DUNSNBR		CHAR(9)
DUNSNBR4		CHAR(4)
CUSTNBR		CHAR(15)
TAXID		CHAR(9)
SOLDATE		CHAR(5)
AWDDATE		CHAR(5)
FAXNBR		CHAR(10)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)

### 3.2.3.73 VARIATIONS Table

Name	Null?	Type
-----	-----	-----
PAPERWORKID	NOT NULL	NUMBER
VARIATIONTYPE	NOT NULL	CHAR ( 2 )
VARIATIONDATE		DATE
ALTERNATEPRICE		NUMBER
UNITPRICECODEBASIS		CHAR ( 2 )
QUANTITY		NUMBER
UNITOFMEASURE		CHAR ( 2 )
VARIATIONDESCRIPTION		CHAR ( 80 )
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR ( 8 )

### 3.2.3.74 VENDOR Table

Name	Null?	Type
-----	-----	-----
VENDORID	NOT NULL	NUMBER
CAGECODE	NOT NULL	CHAR(17)
GOVTPASSWORD	NOT NULL	CHAR(30)
TEMPORARY		CHAR(1)
PREVIOUSVENDORID		NUMBER
INTERCHANGERECEIVERQUALIFIER	NOT NULL	CHAR(2)
INTERCHANGERECEIVERID	NOT NULL	CHAR(15)
APPLICATIONRECEIVERID	NOT NULL	CHAR(15)
ELECTRONICMAILADDRESS	NOT NULL	CHAR(255)
LOCALSYSTEMID		CHAR(7)
DATEASSIGNED	NOT NULL	DATE
PARENTNAME		CHAR(175)
ISAPARENTCOMPANY		CHAR(1)
ISASHELTEREDWORKSHOP		CHAR(1)
ISDEBARRED_SUSPENDED		CHAR(1)
PRISTDINDUSTRIALCLASS		CHAR(30)
OTHSTDINDUSTRIALCLASS		CHAR(30)
NUMBEROFEMPLOYEES		NUMBER
QUOTESMADEASSMALLBUS		CHAR(1)
SITEDesignation		CHAR(17)
ACTIVESTATUS	NOT NULL	NUMBER
STATUSDATE	NOT NULL	DATE
STATUSOPERATION	NOT NULL	NUMBER
STATUSORIGINATOR	NOT NULL	CHAR(8)



---

### 3.2.4 NARQ Code Generation Utility

---

Once the database representation has been formed from the FBI files, a third oraperl script is responsible for parsing the definitions into the C++ source files, C++ header files and single Imakefile needed to generate the NARQ library. The code generator is capable of generating source for all known objects, generating source for a list of known objects and providing a listing of known objects contained in the database representation. It can also manage the output locations for the header files as well as the C++ source files. The Imakefile generated by the script will always generate the rules for the entire library no matter how many objects source is generated for. The primary limitation with the use of an oraperl code generator is also one of its primary benefits. On the plus side, new capabilities can be added by modifications at a single source, the oraperl code generator script. On the minus side, in order to make changes to the source, a fairly steep learning curve is associated with understanding oraperl as well as the structure of the code generation script. Additionally, any software error introduced into the code generator has the potential of being propagated to all NARQ object classes.

---

### 3.2.5 NARQDEF Detail Reference

---

#### 3.2.5.1 DATATYPE Table

Name	Null?	Type
-----	-----	-----
DATATYPEID		NUMBER
FBITYPE		CHAR ( 32 )
DEFINELLENGTH		CHAR ( 1 )
DATATYPEDESCRIPTION		CHAR ( 255 )

### 3.2.5.2 DERIVEDOBJECT Table

Name	Null?	Type
-----	-----	-----
DERIVEDOBJID	NOT NULL	NUMBER
DERIVEDOBJELEMID	NOT NULL	NUMBER
OBJECTIDENTIFIER	NOT NULL	NUMBER
ELEMENTIDENTIFIER	NOT NULL	NUMBER
PUBLICOBJECT		CHAR(1)

### 3.2.5.3 OBJECT Table

Name	Null?	Type
-----	-----	-----
OBJECTIDENTIFIER	NOT NULL	NUMBER
OBJECTNAME	NOT NULL	CHAR(255)
DBTABLENAME	NOT NULL	CHAR(255)
DERIVEDOBJECT	NOT NULL	CHAR(1)

### 3.2.5.4 OBJECTCONSTANTS Table

Name	Null?	Type
-----	-----	-----
CONSTANTIDENTIFIER		CHAR ( 255 )
CONSTANTVALUE		NUMBER

### 3.2.5.5 OBJECTELEMENT Table

Name	Null?	Type
-----	-----	-----
ELEMENTIDENTIFIER	NOT NULL	NUMBER
OBJECTIDENTIFIER	NOT NULL	NUMBER
ELEMENTNAME	NOT NULL	CHAR(255)
DBFIELDNAME	NOT NULL	CHAR(255)
DATATYPE	NOT NULL	NUMBER
LENGTHCONSTANT		CHAR(24)
X12SEGMENT		CHAR(1)
SEQUENCENUMBER		CHAR(8)
DATASIZE		NUMBER
DATAPRECISION		NUMBER
KEYVALUE		CHAR(1)
EXPORTED		CHAR(1)
READONLY		CHAR(1)

### 3.2.5.6 OBJECTRELATIONSHIP Table

Name	Null?	Type
-----	-----	-----
PARENTOBJECTID	NOT NULL	NUMBER
PARENTOBJECTELEMID		NUMBER
CHILDOBJECTID		NUMBER
CHILDOBJECTELEMID		NUMBER
RELATIONID	NOT NULL	NUMBER

### 3.2.5.7 RELATION Table

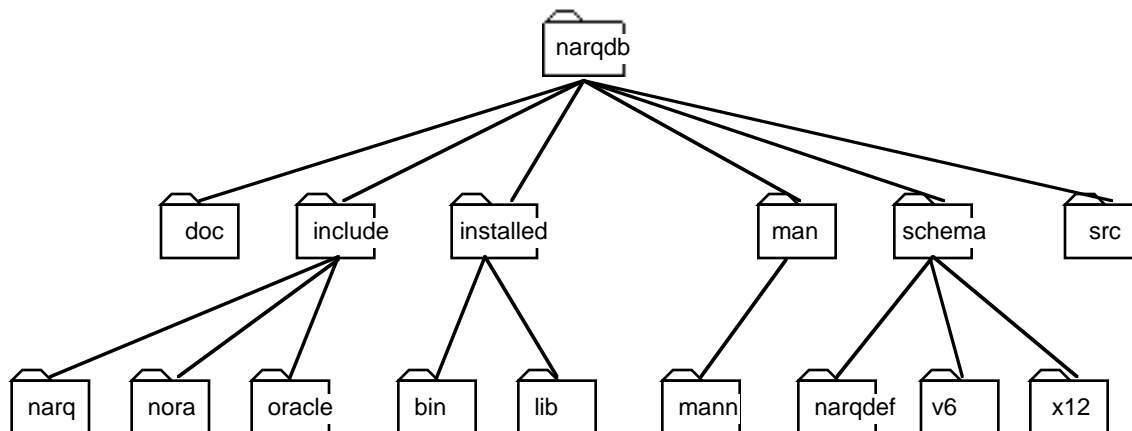
Name	Null?	Type
-----	-----	-----
RELATIONID		NUMBER
RELATIONDESCRIPTION		CHAR ( 255 )

### 3.2.5.8 SIMPLeOBJECT Table

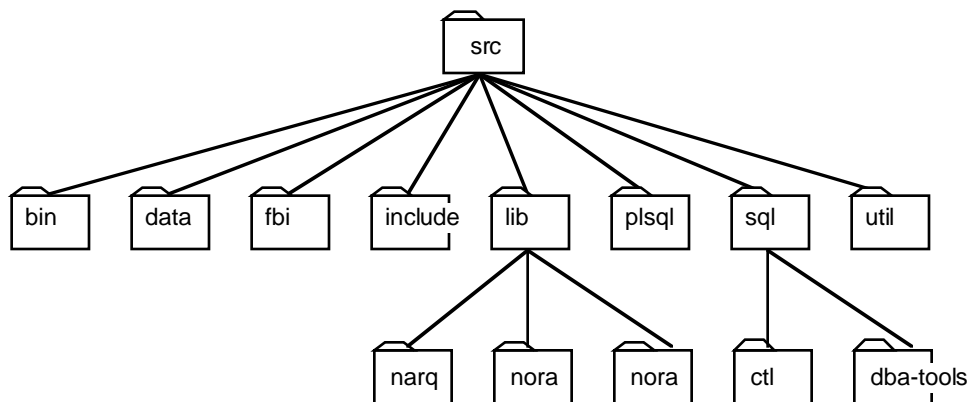
Name	Null?	Type
-----	-----	-----
OBJECTIDENTIFIER	NOT NULL	NUMBER
OBJECTNAME	NOT NULL	CHAR( 255 )
DBTABLENAME	NOT NULL	CHAR( 255 )



At present, the entire source code, database schema descriptions and ASCII documentation is stored in a CVS repository. (*For more information regarding CVS, please refer to on-line documentation*). The current structure of the repository is given in figures 4-1 and 4-2.



*figure 4-1. Structure of /narqdb directory*



*figure 4-2. Structure of /narqdb/src directory*

The procedure used to compile and build the NARQ and NORA libraries is fairly automated. A top-level Imakefile is used to manage the entire generate dependency-compile-link-install process. When moving the code to another machine, there is only one configuration parameter that needs to be set in the Imakefile; the name of the root level directory in which the source exists. For example, if user smith has a home directory of /home/smith in which the NARQ code has been installed into his 'src' subdirectory, the NARQDB variable in the top-level Imakefile would need to be configured as '/home/smith/src/narqdb'. During the development process, it is also necessary to have an environment variable defined the same way. Other tools, notably oraperl scripts, rely on the existence of it.

Building and installing the libraries is a straightforward three-step process. At the root level directory (/home/smith/src/narqdb in our example), type: `xmkmf -a`. (Note: the `xmkmf` script and the `imake` utility that it calls are included with MIT's X11 distribution) This command will read the top-level Imakefile which will instruct it to create Makefiles and associated dependencies for those subdirectories that it has been instructed to step into and that contain Imakefiles. Once this command has stepped through each sub-directory, each subdirectory will contain a Makefile. The second step is to type: `make`. Again, this command will step through each subdirectory following the rules originally defined in each Imakefile. Upon completion, the NARQ, NORA and NARQ\_UTIL libraries will have been built. (The NARQ\_UTIL library is a small set of specialized purpose function calls). The final step will install these new libraries into the top-level installed/lib directory from which they can be moved or copied in order to make them available to other users/developers.

To reiterate, the three steps are as follows:

- 1) Type: `xmkmf -a`  
This will run through each director that has build instructions to defined dependencies and to construct the necessary Makefiles.
- 2) Type: `make`  
This will run through each directory in which a Makefile was constructed in the previous step building the necessary system components.
- 3) Type: `make install`  
This will move the libraries created in the last step and place them in the /installed/lib directory located in the top-level directory.

Changes to the NORA library will either take the form of additional functionality or correction of a programming fault. After making the necessary code changes, the only other steps required are step two and three above. If new object classes are to be added to the NORA library, this will require a change to the Imakefile and a subsequent regeneration of the associated Makefile. It is beyond the scope of this document to explain the format and use of Imakefiles.

Changes to the NARQ library most likely are the result of a change to the GATEC database schema. Because all of the code in the NARQ library is generated from FBI files and is dependent on a consistent database schema, changes to the NARQ library are unnecessarily complicated. The necessary sequence of steps is as follows (for specifics, please refer to the *NORA Design Reference* documentation) :

- 1) modify schema description file and update database schema
- 2) modify FBI textual description and update GATEC schema representation
- 3) generate new C++ object code and associated header file
- 4) build and install NARQ library

The complications surrounding the first step are dependent on whether the table being update currently contains data. If not, the step is straightforward. If so, the existing information must be moved out and reloaded into the new schema. Please refer to the Oracle documentation for details.

The second step is potentially the most difficult because of the lack of tools necessary to facilitate this change. The utility that is used to parse FBI files and maintain the object view of the GATEC schema currently only deals with the addition of columns. It does not handle the removal of columns nor does it handle changes to column type or name. Because of this limitation, it generally recommended that the entire object view be cleaned and reloaded for anything but the addition of new columns.

The generation of the C++ object code is managed by an oraperl script. Because of this, changes to the code can be difficult to make since it is often difficult to determine where the code changes need to be made. During the development process, the normal testing method was to update the code generator, generate a single object (not all objects!) and run tests against just that object before determining whether to apply the changes to the remaining objects.

Because the third step creates a new Imakefile (though it may not differ from the most recent prior version), it is necessary to generate a new Makefile. However, once that is done, it is a simple matter of typing, make, and make install.

---

### 3.4 Database Connection

---

This example is the NORA equivalent of the infamous “hello world” program. The only NORA classes involved are the Database and Connection classes. The program itself attempts to open a connection to a database. It will report back on whether or not it was successful.

```
//
// connect.cc - simple application that attempts to connect
//               to local Oracle database as user 'scott'.
//

#include <stdlib.h>
#include <iostream.h>

#include <nora/Connection.h>
#include <nora/Database.h>

main() {
    Connection* db_connection;
    db_connection = new Connection("scott", "tiger");
    Database* db = Database::instance();
    if (!db->connect(db_connection)) {
        cerr << "Unable to connect to database." << endl ;
    } else {
        cerr << "Connection established!" << endl ;
        db->disconnect() ;
    }

    exit(1);
}
```

The following example codes access a database table called the Buyer table. Its schema description is given below in figure 5-1.

Name	Null?	Type
-----	-----	-----
LocalSystemID	NOT NULL	NUMBER
BuyerID	NOT NULL	CHAR(3)
LastName	NOT NULL	CHAR(35)
FirstName	NOT NULL	CHAR(35)
MiddleInitial		CHAR(1)
PhoneNumber	CHAR(25)	
EMailAddress	CHAR(25)	
LeadStatus	CHAR(1)	
Download	CHAR(1)	

*Figure 5-1. Description of Buyer table.*

---

### 3.4.1 Searching a Single Table

---

The following sample code is the programmatic equivalent to the following SQL statement:

```
SELECT LastName, FirstName, EMailAddress FROM Buyer
WHERE LeadStatus = 'Y'.
```

This sample is meant to introduce the relationship between the Condition and Expression classes as well as the relationship between the SimpleQuery and FetchedRows classes. The code also introduces the Buyer class from the NARQ library as evidenced from the third #include statement. The first part of the code is carried over from the previous example.

In the code, the Condition object is built using a single Expression object. The Expression object specifies the qualifier, "LeadStatus = 'Y'". In turn, the Condition object is used by the SimpleQuery object along with its corresponding placeholder Buyer object. Once the SimpleQuery object is passed to the FetchedRows object, the query is executed - in other words, this is when the database is "hit". Once the query is started, the FetchedRows object controls the retrieval of successive rows as well as determining whether additional data exists. The Buyer object contains the information from each row fetched and using the Column access functions, the database values can be displayed.

```
//
// query_buyer.cc - sample application to query "lead buyer" records from
// Buyer table
//

#include <stdlib.h>
#include <iostream.h>

#include <narq/Buyer.h>

#include <nora/Column.h>
#include <nora/CharColumn.h>
#include <nora/Condition.h>
#include <nora/Connection.h>
#include <nora/Database.h>
#include <nora/Expression.h>
#include <nora/FetchedRows.h>
#include <nora/SimpleQuery.h>

main() {
```

```

//
// Establish connection to the database
//

Connection* gatec = new Connection("scott", "tiger") ;
Database* db = Database::instance() ;
if (!db->connect(gatec)) {
    cerr << "Unable to connect to database." << endl ;
    exit(0) ;
}

//
// Instantiate Buyer object
//

Buyer* buyer = new Buyer() ;

//
// Set up the condition: "Look for lead buyers"
//

Expression* expression = new Expression() ;
expression->compare(buyer->LeadStatusCol(), EQ, "Y") ;
Condition* condition = new Condition(expression) ;

//
// SimpleQuery is used to return information related to a single object
//

SimpleQuery* dbq = new SimpleQuery(buyer, condition) ;

//
// Provide feedback on the number of rows that will be returned
//

cout << "Total number of buyer rows = " << dbq->count() << endl ;

//
// The FetchedRows object actually loads the data into the Buyer object
// and allows iteration through the list
//

FetchedRows* rows = new FetchedRows(dbq) ;

//
// print out all the buyer names
//

while (rows->current() > 0) { // current is negative when error
    cout << buyer->LastNameCol()->value() << ", " ;

```

```

        cout << buyer->FirstNameCol()->value() << " - " ;
        cout << buyer->EMailAddressCol()->value() << endl ;

        rows->next() ; // fetch the next row of information
    }
    //
    // disconnect from the database
    //

    db->disconnect() ;

    // release the storage for the allocated components

    delete expression ;
    delete condition ;
    delete dbq ;
    delete rows ;
    delete buyer ;
}

```



The following sample code will create a new record in the Buyer database table. The only new object in this sample code is the Sequence class. The Sequence object is used to derive a unique identifier value for the new record. It is worthwhile to note how values are “assigned” to columns. They can be assigned by function call, but most derived Column classes have an assignment operator defined. Hence, the following two calls are equivalent:

```
Table->Column("value"); // function call assignment
Table->Column = "value"; // overloaded assignment
```

Lastly, after a record has been defined, calling the commit() member function of the Buyer table object will insert the record into the database. However, the change is not permanent until the commit() member function of the appropriate Database or Connection object is called. Similarly, to cancel the record insertion, the rollback() member function of the appropriate Database or Connection object must be called.

```
//
// create_buyer.cc - simple application to create a new record in the
// Buyer table
//

#include <stdlib.h>
#include <string.h>
#include <iostream.h>

#include <narq/Buyer.h>

#include <nora/Column.h>
#include <nora/CharColumn.h>
#include <nora/Connection.h>
#include <nora/Database.h>
#include <nora/NumberColumn.h>

main() {

    //
    // Establish connection to the database
    //
    Connection* gatec = new Connection("scott", "tiger");
    Database* db = Database::instance();
    if (!db->connect(gatec)) {
        cerr << "Unable to connect to database." << endl;
    }
}
```

```

        exit(0) ;
    }

    //
    // Create a new buyer for insertion
    //

    Buyer* buyer = new Buyer() ;

    Sequence* seq = new Sequence("seq_BuyerID") ;

    buyer->LocalSystemIDCol(seq->next_value()) ;
    buyer->BuyerIDCol("000") ;
    buyer->LastNameCol("Doe") ;
    buyer->FirstNameCol("Jane") ;
    buyer->PhoneNumberCol("5105551212") ;
    buyer->EmailAddressCol("jd@anonymous.com") ;
    buyer->LeadStatusCol("Y") ;

    if(buyer->commit()) {
        cout << "Buyer 000 (Jane Doe) added to the Buyer
database" << endl ;
    } else {
        cerr << buyer->error_msg() << endl ;
        cerr << buyer->ora_error_msg() << endl ;
    }

    //
    // disconnect from the database
    //

    db->disconnect() ;

    delete buyer ;
    delete seq ;
}

```

This final sample code is virtually identical to the `buyer_query.cc` sample code presented earlier. The differences in the following sample are manifested in a change to the prepared query and to the loop that returns the rows returned by the query. The query has been modified to look for “anonymous” lead buyers. In this case, anonymous buyers are those defined as having a `buyerid` value of “000”. As each row is returned from the query, the `delete_row()` member function of the `Buyer` object is called. This effectively removes the row once the `commit()` member function of appropriate Database or Connection object is called.

---

### 3.4.3 Deleting an Existing Record

---

```
//
// buyer_delete.cc - simple application to remove (tag) a record from the Buyer table
//

#include <stdlib.h>
#include <iostream.h>

#include <narq/Buyer.h>

#include <nora/Column.h>
#include <nora/CharColumn.h>
#include <nora/Condition.h>
#include <nora/Database.h>
#include <nora/Expression.h>
#include <nora/FetchedRows.h>
#include <nora/SimpleQuery.h>

main() {

    //
    // Establish connection to the database
    //

    Connection* gatec = new Connection("scott", "tiger") ;
    Database* db = Database::instance() ;
    if (!db->connect(gatec)) {
        cerr << "Unable to connect to database." << endl ;
        exit(0) ;
    }

    //
    // Instantiate Buyer object
    //

    Buyer* buyer = new Buyer() ;

    //
    // Set up the condition: "Look for Lead buyers"
    //

    Expression* expression = new Expression() ;
    expression->compare(buyer->LeadStatusCol(), EQ, "Y") ;
    Expression* expression_2 = new Expression() ;
    expression->compare(buyer->BuyerIDCol(), EQ, "000") ;

    Condition* condition = new Condition(expression) ;
```

```

condition->and(expression_2) ;

//
// SimpleQuery is used to return information related to a single object
//

SimpleQuery* dbq = new SimpleQuery(buyer, condition) ;

//
// Indicate the number of rows to be removed
//

cout << "Total number of rows to be deleted = " << dbq->count() << endl << endl ;

//
// The FetchedRows object actually loads the data into the Buyer object
// and allows iteration through the list
//

FetchedRows* rows = new FetchedRows(dbq) ;

//
// print out all the buyer names
//
while (rows->current() > 0) { // current is negative when error
    cout << buyer->LastNameCol()->value() << ", " ;
    cout << buyer->FirstNameCol()->value() << " - " ;
    cout << buyer->EmailAddressCol()->value() << endl ;

    cout << endl << buyer->contents(false) << endl ;

    // remove anonymous (Jane Doe) buyer
    buyer->remove_row() ;

    rows->next() ; // fetch the next record
}

//
// commit the change
//
db->commit() ;

//
// disconnect from the database
//
db->disconnect() ;

// release all of the storage for these components

delete expression ;

```

```
delete expression_2 ;  
delete condition ;  
delete dbq ;  
delete rows ;  
delete buyer ;  
}
```

**Embedded-SQL** is a mechanism for including SQL calls in a higher-level programming language. Currently, Oracle interfaces exist for C, COBOL, FORTRAN and Ada. The lack of a C++ interface and the performance advantage of using lower level OCI calls led to the abandonment of its inclusion.

**FBI** This is an acronym for **F**ield **B**inding **I**nterface. Referring to an FBI usually implies a reference to the textual description of a GATEC database object. FBI files are pre-processed into a database format for later use by a C++ code generator.

**Imakefile** A template that is used to generate a Makefile. This allows machine dependencies to be kept separate from the various items that need to be built.

**NARQ** This is an acronym for **N**otes, **A**cquisitions, **R**equests for quotes and **Q**uotes. It refers to the C++ object library containing Oracle-specific access routines.

**NORA** This is an abbreviation for **NARQ ORACLE**. It refers to the C++ object library containing GATEC components.

**OCI** The **O**racle **C**all **I**nterfaces are a set of lower-level C libraries upon which the NORA library is constructed. The advantage of OCI is that it leaves the issue of cursor management to the programmer.

**Oraperl** Perl access to Oracle databases. See perl.

**Perl** Perl is an interpreted language optimized for scanning arbitrary text files, extracting information from those text files, and printing reports based on that information. This utility was written by Larry Wall and is available from many internet on-line services.

---

---

## SECTION 4 CDFDB Library

---

The CDFDB library is a set of C++ routines/classes which interfaces with the Oracle Database. Its main purpose is to handle incoming and outgoing X.12 transactions which have been translated into a Common Data Format (CDF). For X.12 transactions going from the GATEC 2 System to the VAN Hub, CDF files are generated for each transaction type, queued for translation to X.12 and then sent via electronic mail to the hub. For X.12 transactions going from the VAN Hub to the GATEC 2 System, the X.12 transaction is received by an inbound Bourne Shell script, translated to a CDF, processed by the appropriate CDFDB application and inserted into the database. A transaction flow diagram is shown in Figure 1.

---

### 4.1 Design Intent

---

The design of the CDFDB library was targeted at bridging the gap between an X.12 transaction translated into a CDF file and the NARQ object library which is the database Application Programming Interface (API) used in the GATEC 2 System. The NARQ library was designed and developed in parallel with the GATEC 1 Interim System, put into production in August 1992. GATEC 1 used the Government Standard Translator (GST) for translating X.12 documents to and from a CDF file for easy data manipulation. Files were used to store transaction data instead of a database. The use of the GST proved to be invaluable in GATEC 1 because it freed the application programmer from having to parse X.12 and it put the data in a more desirable format. Continued use of the GST seemed quite in order for GATEC 2. However, its use created a new problem since the NARQ library was designed and developed independently from the GST. There was a need to pipe output from the GST to the database and pipe database output to the GST. It was then determined that the continued use of the CDF was also in order despite some overhead disadvantages. A great advantage to this approach, however, was that it was proven and it did lend itself rather nicely to handling incoming transactions (discussed later).

---

## 4.2 Dependencies

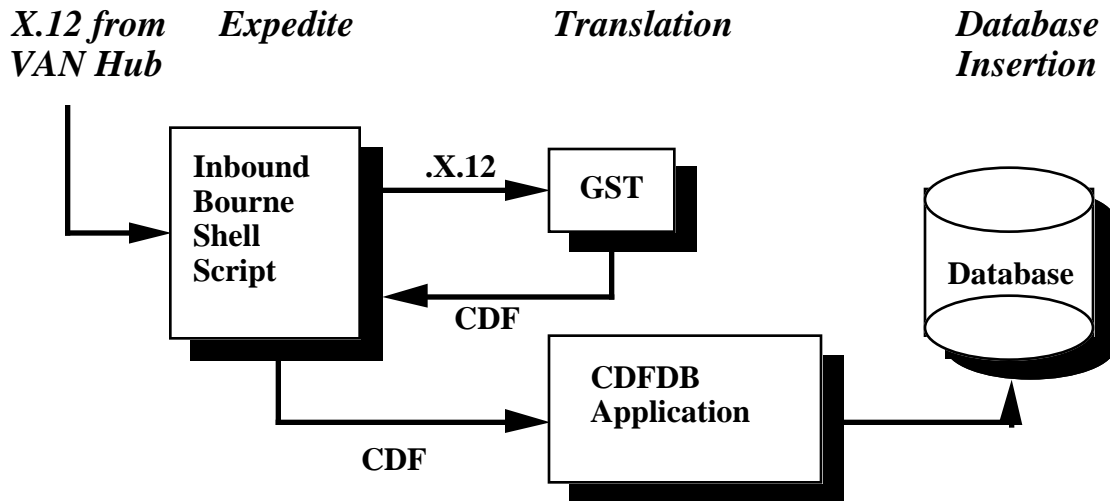
---

The CDFDB library is extremely interdependent on other libraries and software modules. Remember its design was to act as a bridge between the GST and the database. Using it for any other purpose other than the aforementioned renders it completely useless. In order for the library to compile, it first needs a C++ compiler since the code is written in C++. It is highly recommended that a Sun C++ compiler on SunOS be used to build the library because it has not been ported nor tested on any other platform.

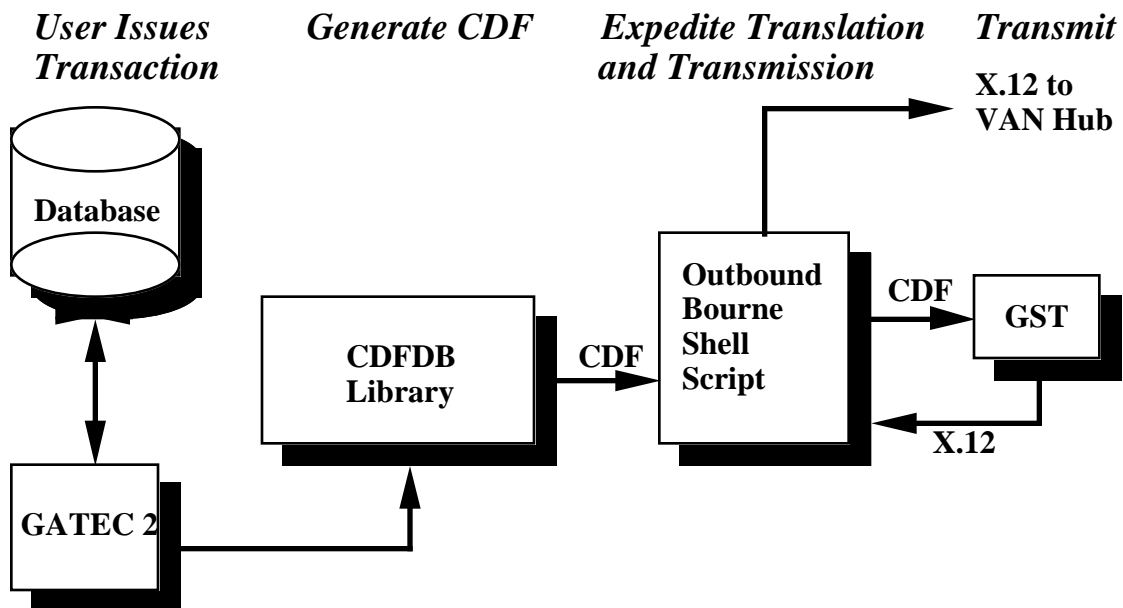
To build a useful application such as GATEC or 843CDFtoDB for example, several libraries need to coexist in order for the link to be successful. The Oracle libraries which come with the Oracle System are probably the most important. Every library or software module that acts as a client to the database depends on these libraries.



### Inbound Flow Diagram



### Outbound Flow Diagram

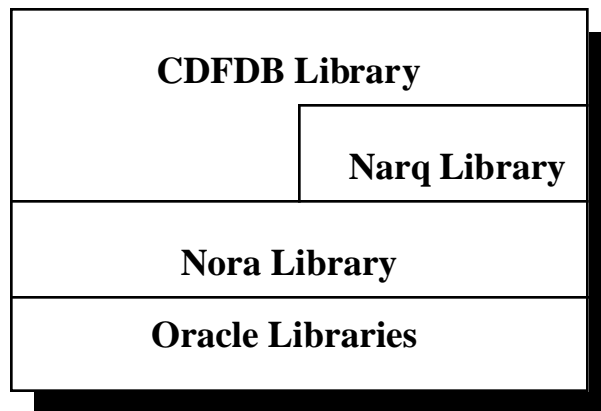


**Fig. 1 Inbound and Outbound Flow.**

The libraries are listed below:

```
$(ORACLE_HOME)/rdbms/lib/libora.a
$(ORACLE_HOME)/rdbms/lib/libsqlnet.a
$(ORACLE_HOME)/rdbms/lib/liboci14c.a
$(ORACLE_HOME)/proc/lib/libc14.a
$(ORACLE_HOME)/proc/lib/libcgen.a
$(ORACLE_HOME)/rdbms/lib/osntab.o
```

The next set of libraries which CDFDB depends on is the NARQ and NORA libraries. These libraries are the object oriented front end to the Oracle Database libraries. They were written not only to give the application programmer an object oriented view to the relational database, but to also simplify data insertion/retrieval and to minimize requirements to change application code when changes were made to the schema. NORA contains base classes for tables, columns, and queries. The NARQ library contains derived classes for specific tables and columns in the schema. All calls made by CDFDB are to both NARQ and NORA. None are made directly to the Oracle libraries. The call hierarchy is illustrated below:



**Fig. 2 CDFDB call hierarchy**

For further details of Oracle, NORA and NARQ libraries, refer to the database section of the GATEC 2 Internal Description and Maintenance Guide.

The last library needed for a successful build is the tispq (queue) library. This library contains routines which allow a programmer to queue CDF's to be processed. These calls are made when a CDF is waiting to be uploaded to the Legacy System (discussed later) and translated by the GST. For further detail on the tispq library and the applications created to monitor a specific queue, refer to the Queuing section of GATEC 2 System manual.

After the CDFDB library is compiled and linked properly with the application software, it relies on the GST for its translation capabilities. Since CDFDB only understands CDF formatted data, it depends on the GST to translate CDF's to X.12 on outgoing transactions and X.12 to CDF's on incoming transactions. Without the GST, transactions could not leave nor enter the GATEC system.

---

### 4.3 Advantages to the CDF Approach

---

There are many advantages to using a CDF for data processing. The most obvious is its ease of readability. X.12 is very cryptic and reading it requires an expert or always having an X.12 Standards volume at hand. With the use of a CDF, transaction data is laid out in a simple, easy to read format. Each bit of transaction data is labeled with the table name and the column name of where this data maps to in the database and since the tables have been designed to be self documenting, transaction data is more comprehensible by the reader. This is especially useful for tracing transaction flow with the system and debugging programs. The format of a CDF line is as follows:

`%table_name.column_name transaction_data`

Also, what's easier to read by a human is also easier to read by a program. With the CDF, further translation is not required. The computer process reads the table name and column name and knows immediately where the data is to be inserted in the database. This is because of NARQ's unique way of storing a name with each column and table class.

Another advantage to using CDF's is that it removes the complexity of X.12 out of the application program and into the GST which was specifically designed for X.12 translation purposes. The GST has a workbench which allows the user to create and watch translations take place (WYSIWYG) and when completed, the translation can be called by other processes. No programming in a traditional sense is required. This frees the application programmer >from having to do tedious X.12 parsing. It also makes the GATEC system a bit more modular.

---

#### 4.4 Disadvantages to the CDF Approach

---

Although there are many advantages to the CDF approach, there are also some disadvantages that one must consider. One is the overhead costs of using GST. Every time a transaction is translated, there are start-up costs involved which puts a bottleneck in the transaction delivery. An alternative would be to keep GST running as a daemon and having a front end server to the GST, however it would require additional software. There are also overhead costs associated with using NARQ. The use of NARQ tends to make gigantic executable files which can create a problem if disk space is short or the executables need to be shipped across the Internet. Another disadvantage is the tremendous amount of coordination required between both the application programmer and the person creating the translation. Often there are disagreements with what's expected by the CDFDB process and the GST on required data elements, data names and content.

Overall, the advantages of the CDF approach outweigh the disadvantages because of the mere fact that it caters more to people who must maintain the system.

---

#### 4.5 Short Comings (Implementation)

---

Cannot handle multiple transactions within a CDF.  
Does not check data for proper format.

---

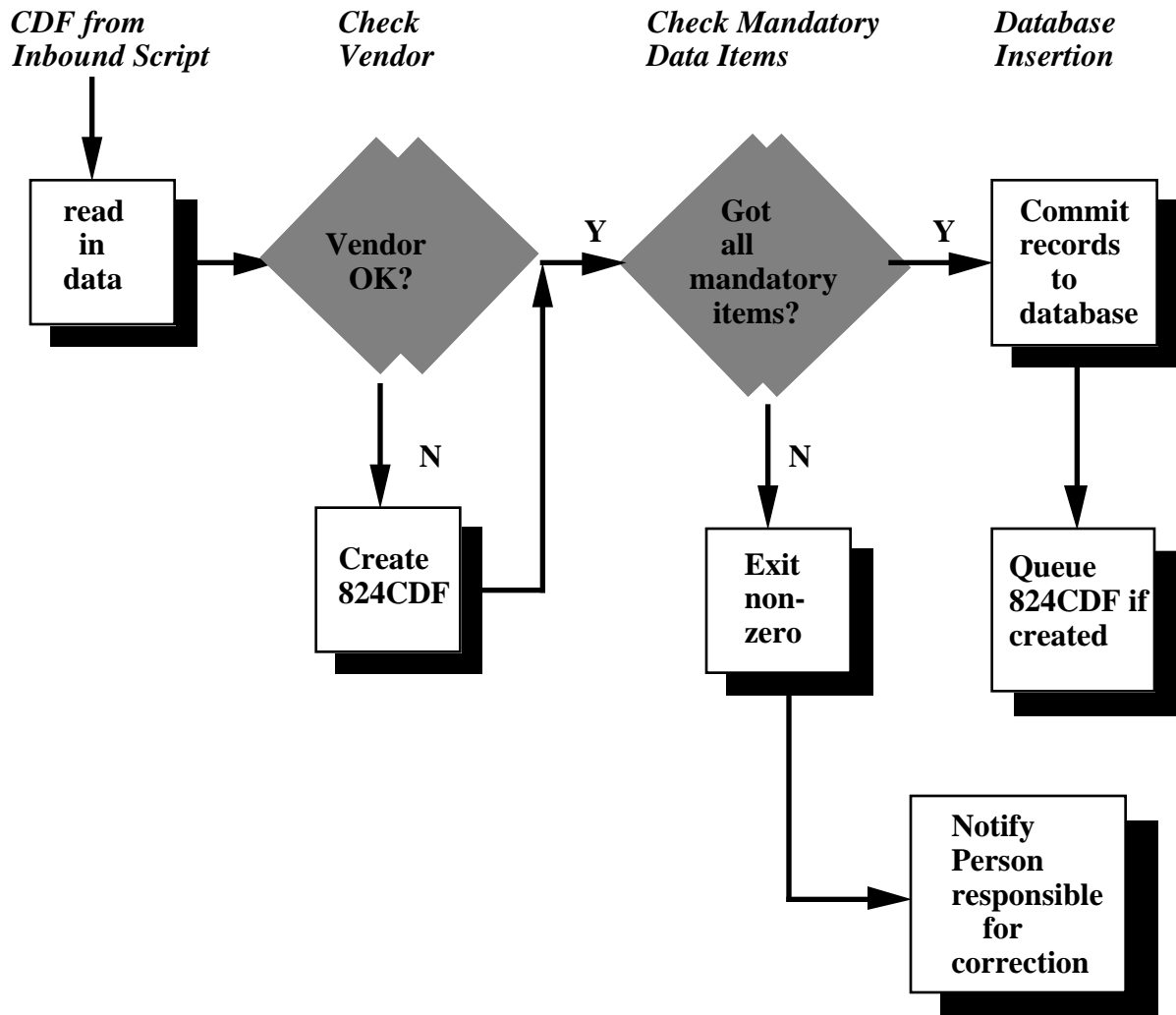
#### 4.6 CDFtoDB

---

The purpose of the CDFtoDB class is to process incoming CDF's and insert the data in the database (hence the name). Part of the process involved is vendor validation. For each CDF transaction that is fed to CDFtoDB, the vendor cagecode and government password is checked against vendor records. If the transaction fails the validation process, CDFDB generates an 824CDF (Application Advice) for vendor notification and sends it to the GST for outbound delivery. If vendor validation succeeds, CDFDB checks for mandatory data elements for a particular transaction type. If mandatory data elements were missing from the CDF or there were problems committing records to the database, CDFDB will exit with a non-zero exit status and the inbound script will deliver via email the errors found with the offending CDF and the original

X.12 transaction to the programmer(s) and system administrator(s) responsible for correction. A flow diagram is shown in Figure 3.

### CDFtoDB Flow Diagram



**Fig. 3 CDFtoDB flow.**

NAME

chk\_mand

SYNOPSIS

```
class chk_mand
{
    string tbl_name;
    CM_ENTRY *cm_tbl;
    int size;
    int tbl_index;
    string status_str;
    string ref_tbl_name;
    CDFBCAS_DB_REF_ENTRY *ref_tbl;
    int ref_tbl_size;
    string ref_tbl_index;

public:
    chk_mand(string, CM_ENTRY*);
    chk_mand(string, CM_ENTRY*, string, CDFBCAS_DB_REF_ENTRY*);
    int find_item(char*) const;
    int set_itemok(char*);
    int set_itemok(int);
    int is_ok();
    void set_status(char*);
    char *get_status() const;
    char *get_next();
    CDFBCAS_DB_REF_ENTRY *get_next_ref_ent();
    int get_index() const;
    string get_tbl_name() const;
    string get_ref_tbl_name() const;
    char *name_to_dbname(char*) const;
};
```

SYNOPSIS

```
#include <stdio.h>
#include <iostream.h>
#include "common.h"
#include "chk_mand.h"
#include "cdfdb.h"
```

```
int CDFtoDB(Database *db, TBL_PTR_ENTRY *tbl_ptrs,
            int tbl_size, short doctype, chk_mand *cm_obj)
```

## DESCRIPTION

CDFtoDB() expects five arguments. The first argument is a pointer to the database class. It assumes that a successful connection has been made to the remote database server. For further discussion on the Database class, refer to the NORA library section of the GATEC 2 System manual. If the connection has not been made, it will return a -1 to the calling routine because of failure to get a sequence number from the database. The message it will write to standard error is as follows:

```
fill_docids(): unable to get DocumentId sequence
CDFtoDB(): fill_docids() failed
```

The second argument is a pointer to a table or list of table pointer entries. CDFtoDB() assumes at least one table entry that has been initialized. Generally, there will be more than 1 table because there's been great effort in the GATEC database design to keep transaction data normal. The third is the number of table entries in the list. If there are zero entries, the process will produce unexpected results. The fourth is the document type (i.e. 843, 864, etc.). Unfortunately, CDFtoDB() needs to know which document type its inserting so it knows to bypass vendor validation on transactions not requiring it (i.e. 824's and 838c's). The last argument is a pointer to the chk\_mand object (class declaration shown above). This object contains a list of data items that are determined to be mandatory for a particular transaction type. CDFtoDB() checks off items that have been found in the document read in. If any of the items have not been checked off, it will display which data items are missing and return -1 to the calling routine.

## ERRORS

CDFtoDB() returns OK or 0 if the CDF has been processed successfully.

On failure, CDFtoDB() will return one of the following:

ERR            General programming error (-1).

RECOMMIT\_ERR   Record Commit Error. This will occur when there is a failure on the first phase commit or when there are mismatching ids (3).

DBCOMMIT\_ERR   Database Commit Error. This will occur when there is a failure on the second phase commit (4).

CAGECODE\_ERR   CageCode error. This will occur when the cagecode specified in the transaction could

not be found in the database (6).

**GOVTPASSWORD\_ERR** Government Password error. This will occur when there was an incorrect government password given with the cagecode specified in the transaction (7).

**NOVENDOR\_ERR** No Vendor error. This will occur when the Vendor table was not found in the list of tables (8).

**NOVALIDINFO\_ERR** No Valid Information error. This will occur when both Cagecode and Government password are incorrect (9).

---

## 4.8 Creating New Applications for New Document Types

---

Creating a CDF insertion program for a new transaction type is quite easy, assuming that the tables have been created in the database and the NARQ code has been generated to accommodate them. The best approach would be to modify an existing program such as 843CDFtoDB.cc in this document. Writing a program from scratch can be very time consuming and prone to error. An advantage to using the CDFtoDB() is that its been in production for nearly 1 year and most if not all bugs have been addressed.

To create the program using CDFtoDB(), you need to first include the .h files from NARQ that represent the tables used. For example, lets say we're trying to create a program to process 810 CDF's. If the CDF contains data for the Contact table, Document table, etc., then you need to include them at the top of your .cc file:

```
#include <narq/Contact.h>
#include <narq/Document.h>
#include <narq/FreeOnBoard>
#include <narq/OriginalTransaction>
#include <narq/ShippingDocPackage>
#include <narq/TransactionSent.h>
#include <narq/Vendor.h>
```

Next, you need to list all the data items that are required by the database. So for example, if the CageCode and GovtPassword fields of the Vendor table are required, you would have the following as your status table declaration:

```
// Status table declaration
```



```

CM_ENTRY CDF810_status[] =
{
    { 0, "Vendor.CageCode", 0 },
    { 0, "Vendor.GovtPassword", 0 },
    { EOT, "", 0 }
};

```

The next step would be to instantiate all the objects used by the program. Each table object instantiated would be pointed to by `tbl_ptr` within the `TBL_PTR_ENTRY` struct:

```

typedef struct
{
    boolean modified;    // Set when modified by incoming data.
    boolean hold;        // Set when table should not be cleared.
    boolean commitable;  // Set when allowed to be committed to
the database
    Table* tbl_ptr;
}TBL_PTR_ENTRY;

```

The modified flag is set when data has been read in and put into the table object pointed to by `tbl_ptr`. The hold flag is defaulted to FALSE but should be set to TRUE if you wish for `tbl_ptr` to be static (not cleared by `clear_tables()`). The commitable flag is defaulted to TRUE, however, if you want the table as just a place holder for data and not a record to commit to the database, set it to FALSE.

// Instantiate all the necessary table objects

```

tbl_ptrs[0].tbl_ptr    = new Contact();
tbl_ptrs[1].tbl_ptr    = new Document();
tbl_ptrs[2].tbl_ptr    = new FreeOnBoard();
tbl_ptrs[3].tbl_ptr    = new OriginalTransaction();
tbl_ptrs[3].hold       = TRUE;
tbl_ptrs[3].commitable = FALSE;
tbl_ptrs[4].tbl_ptr    = new ShippingDocPackage();
tbl_ptrs[5].tbl_ptr    = new TransactionSent();
tbl_ptrs[5].hold       = TRUE;
tbl_ptrs[5].commitable = FALSE;
tbl_ptrs[6].tbl_ptr    = new Vendor();
tbl_ptrs[6].hold       = TRUE;
tbl_ptrs[7].commitable = FALSE;

```

Since `tbl_ptrs` is an array, the size of it needs to be set. In this case, it would be set to 8. Please make sure its the right size because it can be easily overlooked. The entire program depends on the proper list size and getting it wrong can cause all kinds of havoc.

```

const int  tbl_size = 8;

```

Also, we want to set doc\_type to the proper document type:

```
short      doc_type = 810;
```

And that's it. There may be a few exceptions depending on the transaction type but for the most part, the programming is complete. If you used an existing CDFtoDB program as a template, you may want to change all references to the old transaction type (i.e. "843" to "810").

---

## 4.9 Existing Applications

---

The existing applications which use CDFtoDB() are 843CDFtoDB for handling Responses to Request for Quotations (843), 864CDFtoDB for handling incoming Text Messages (864), 838cCDFtoDB for processing Trading Partner Profile Confirmations (838c) for registering vendors at the site, and 824CDFtoDB for handling Application Advice transactions (824). All programs are very similar with a few small exceptions. They all read stdin for input, connect to remote database server and have two command line options. The options are as follows:

### OPTIONS

- t Puts program trace on for debugging purposes.
- p Sets the production flag on so outgoing transactions get queued for translation and mailed.

### FILES

~gatec2/etc/dblogin      to get login info for active database.

To help the reader get a better feel of these programs, comments on each are provided below:

---

### 4.9.1 843CDFtoDB

---

This is by far the popular program since there are hundreds of Responses to Request for Quotes coming into the GATEC system each day. It is also the most generic and probably the best example to use for creating your own insertion program.

---

#### 4.9.2 838cCDFtoDB

---

This program is used occasionally when a vendor registers. CDFtoDB() knows not to check for cagecode and government password since it is a vendor registration program. The only other exception is the logic used to set Vendor.TemporaryCol and Vendor.DateAssigned.

---

#### 4.9.3 824CDFtoDB

---

824CDFtoDB is another fairly generic program. CDFtoDB() also knows not to check for cagecode and government password because its not really necessary on Application Advice. It wouldn't be bad to use 824CDFtoDB as an example or a template for creating your own program.

---

#### 4.9.4 864CDFtoDB

---

864CDFtoDB would appear to be quite similar to the other programs but in actuality, it's a bit different. There is some special handling for 864CDF's which can be found in the readin\_data() routine. Since an 864 can have looping segments, multiple records of the same type needed to be inserted into the database. In order to handle it, readin\_data() needed to do intermittent phase 1 commits. When a table has been completely read in, the 864CDF will typically have a tablename.commit line which will follow the data. This tells the CDFtoDB program to issue a phase 1 commit. For example, if you had the following multiple segments in your 864 transaction:

```
PER*IC* DEWEY STAATZ*TE*(719) 473-8896
PER*IC* DEWEY STAATZ*TE*1-800-359-4157
PER*IC* DEWEY STAATZ*FX*(719) 632-7900
```

The GST would translate it into the following CDF format which contains the intermittent commits:

```
%Contact.Name MARK N. LYNCH
%Contact.PreferredAccess TE
%Contact.PhoneNumber 919-483-1212
%Contact.commit
```

```
%Contact.Name MARK N. LYNCH
%Contact.PreferredAccess FX
%Contact.FaxNumber 919-483-4083
%Contact.commit
```

The CDFtoDB program would do the commit on the first occurrence, clear out the memory, read in the next record and do the commit. If all phase 1 commits were successful, the program will do the second phase commit. Otherwise, the program would do a rollback and exit with the appropriate error code.

---

#### 4.10 Short Comings

---

New program has to be created for each document type.

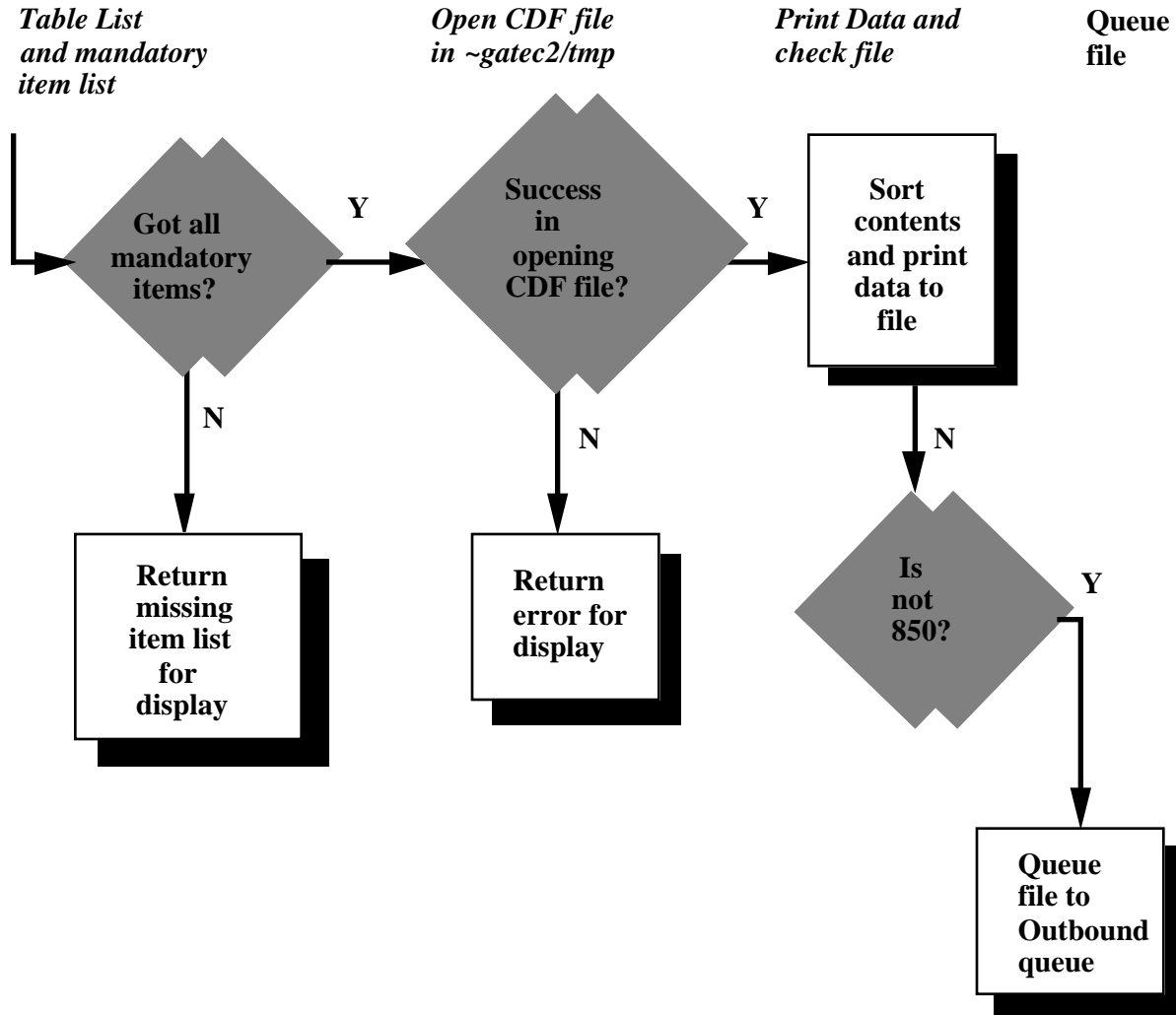
---

#### 4.11 DBtoCDF

---

The purpose of the DBtoCDF() class is to generate CDF's for outbound translation and delivery. DBtoCDF gets executed when a user issues Request for Quotes (RFQ) or Awards to be delivered to a VAN. DBtoCDF assumes a list of table objects with data and a list of data elements that are mandatory in a particular X.12 transaction. Upon execution, DBtoCDF checks the table list against the mandatory data element list. If any elements are missing, CDFtoDB returns the missing item list to the calling routine so it can be displayed to the user. If all mandatory data elements are present, DBtoCDF opens a temporary file in ~gatec2/tmp. The name of the temporary file is determined by the CDF type (i.e. 840ZAAa09485 for 840's, 850XAA12213 for 850's). If all is well with opening the file, DBtoCDF will print special headers (shown below) at the beginning of the file to further describe it. The outbound delivery script uses these headers to determine which CDF to X.12 translation to use. After printing the headers, DBtoCDF begins printing the contents of each table to the file. While printing, it sorts each table by column name. When completed, DBtoCDF will queue the CDF file for delivery but only if the CDF is not an 850CDF. If it is an 850 CDF, it is put on a different queue (discussed below). DBtoCDF flow is shown in Figure 4.

### DBtoCDF Flow Diagram



**Fig. 4. DBtoCDF flow.**

NAME

DBtoCDF()

SYNOPSIS

```
#include <iostream.h>
#include "common.h"
#include "chk_mand.h"
#include "cdfdb.h"

DBtoCDF(chk_mand *cm_obj,
        char      *file_template,
        char      *hdrs,
        TBL_PTR_ENTRY *tbl_ptrs,
        int       tbl_size,
        short     doctype,
        char      *order_statements)
```

DESCRIPTION

DBtoCDF() expects 7 arguments. The first argument is a pointer to the chk\_mand object which must be instantiated in the calling routine. This object will contain a list of mandatory data elements for a particular transaction type and member functions to update a check list. The file\_ template argument is the name of the temporary file created. The name will typically be the type of CDF (i.e. "840", "850"). The hdrs argument is the header string which the caller wishes to put at the head of the CDF file created. A typical header for a CDF will look like this:

```
%Xbegin
%Xpurpose      Award
%Xfilename     850CDF
%Xdestination_host translator
%Xversion      2
%Xdate         93 07 14
```

The outbound delivery script uses the %Xpurpose line to determine which CDF to X.12 to use. The fifth argument, tbl\_size, is the number of tables in the list. Its very important that this argument is correct since nearly all the DBtoCDF code depends on it. Doctype is the type of document (i.e. 840, 850), and order\_statements is the statement of order declaration. Order\_statements typically appear in an 850. If order\_statements are not needed, DBtoCDF will accept a NULLSTR for in its place.

Having front end routines to DBtoCDF() which accept pointers to individual table objects were originally designed to let the GATEC application programmer know what table objects were expected for a particular transaction type. They are not necessary since DBtoCDF() can be called directly, however, it is recommended that this approach be taken to minimize table type confusion and to better associate database tables with transactions sent to VAN's. The best way to create a new routine is to use one that already exists. 840DBtoCDF() would be a good example. Let's say we're trying to send an 810 transaction this time. We would include the .h files representing each table object in our 810DBtoCDF.cc file and the list of required data elements for the 810 transaction. The EOT at the end of the CDF810\_status list signifies the end of table.

```
#include <narq/Contact.h>
#include <narq/Document.h>
#include <narq/DocumentSent.h>
#include <narq/GSDefaults.h>
#include <narq/ISADefaults.h>
#include <narq/FreeOnBoard>
#include <narq/ShippingDocPackage>

// Status table declaration

CM_ENTRY CDF810_status[] =
{
    { 0, "FreeOnBoard.FOBType", 0 },
    { 0, "FreeOnBoard.FOBDescription", 0 },
    { 0, "FreeOnBoard.FOBAcceptancePoint", 0 },
    { 0, "FreeOnBoard.FOBAlternateInspection", 0 },
    { 0, "FreeOnBoard.FOBInspectionPoint", 0 },
    { 0, "ShippingDocPackage.DocDeliveryMethod", 0 },
    { 0, "ShippingDocPackage.DocumentType", 0 },
    { EOT, "", 0 }
};
```

The function prototype would look like this:

```
int _810DBtoCDF(chk_mand* cm_obj,
                Contact *con,
                Document *doc,
                DocumentSent *ds,
                GSDefaults *gsd,
                ISADefaults *isad,
                ShippingDocPackage *sdp)
```

Notice how everything above is sorted. This helps in making sure all the necessary components are included in this .cc file. It also helps the GATEC application programmer in making sure the call to this routine has all the necessary arguments and that they're in their proper order. Putting each argument on a separate line helps the reader easily know what's expected. It's probably a good idea to mention in a comment block that your routine expects instantiated objects with transaction data in them. If any of objects in the parameter list is not instantiated, DBtoCDF() will most likely core dump. If you have an instantiated object passed to your routine, but there is no data, only the table name will get printed to the CDF. The next step would be to store each table in an array since DBtoCDF() works with the array (or list of table objects) to create the CDF. Its very important that the numbers (i.e. the size of tbl\_ptrs and the indices upon assignment) are correct.

```
string          hdrs;
TBL_PTR_ENTRY  tbl_ptrs[6];
char           cdf_filename[MBUFSZ];
short          doctype = 810;
```

```
// This way is acceptable by CC and gcc
```

```
tbl_ptrs[0].tbl_ptr = con;
tbl_ptrs[1].tbl_ptr = doc;
tbl_ptrs[2].tbl_ptr = ds;
tbl_ptrs[3].tbl_ptr = gsd;
tbl_ptrs[4].tbl_ptr = isad;
tbl_ptrs[5].tbl_ptr = sdp;
```

Again, if you used an existing DBtoCDF routine, you may want to change all document references from the old document type to the new document type (i.e. "840" to "810").

---

#### 4.13.1 Applications using DBtoCDF

---

GATEC so far is the only application that uses DBtoCDF routines. When a buyer issues an RFQ from the Review RFQ screen, 840DBtoCDF() is called to create the 840CDF to be queued for outbound delivery to the VANs. Only when a CDF has been successfully created and queued will GATEC commit its records to the database. Otherwise, the user will be notified of the problem in issuing the transaction. When the buyer issues an award from the Award screen, 850DBtoCDF() gets called. This front end routine creates an 850CDF for awarding the winning bidder, an 836CDF for notifying the PUBLIC of the winning bidder, and calls



BCASCDFtoDB() to create a BCASCDF for updating BCAS (BCASDBtoCDF explained below). 850DBtoCDF() is a special case routine which diverges from the normal DBtoCDF front end. These three DBtoCDF routines were the only ones used in the production system. 864CDFtoDB() was not used in GATEC for reasons explained in the DBtoCDF Short Comings section of this document.

To help the reader get a better feel of these routines, comments on each are provided below:

---

#### 4.13.1.1      840DBtoCDF

---

This routine is very generic and would be a good example to follow if you needed to generate another CDF type. The call requires that all objects be instantiated and has data in them. The programmer is responsible for deleting the objects after the CDF has been generated. An example of the call can be found in:

~user/dui/src/applications/applications/gatec/Review\_RFQ.C

---

#### 4.13.1.2      850DBtoCDF

---

This routine is sort of a three in one routine. In other words, 3 CDF's get created from one set of Award tables. This is a special case routine so it is probably not a good idea to use if you are writing your own CDF generation routine. 850CDFtoDB() will create 3 CDF's (850, 836, BCAS) as long as its not a transaction cancel. If it is a transaction cancel, only the 850 and 836 CDF's get created because the BCAS cancel CDF was already generated by the caller. In both cases, nonetheless, queadditem() gets called which handles monitoring whether or not the BCASCDF upload was successful. The 850CDF transaction only gets put on the outbound queue if the upload was successful. For further details of how queadditem() operates, refer to the "q" man page. An example of the call can be found in:

~user/dui/src/applications/applications/gatec/Award.C

---

#### 4.13.1.3 BCASDBtoCDF

---

This routine is called by 850CDFtoDB() to generate the BCASCDF for uploading Award information to BCAS. It made sense to have 850CDFtoDB() call it because it uses a subset of the database tables used to generate the 850CDF and it only required one call made by the GATEC application. Since the BCASupload script uses a GATEC 1.0 formatted CDF, BCASDBtoCDF() uses a cross reference table for its translation. This was done so the BCAS upload script did not require changes to accommodate the new CDF format (introduced in GATEC 2.0). The cross reference table can be found in BCASCDFtoDB.cc.

---

#### 4.13.2 DBtoCDF Short Comings

---

The DBtoCDF routines do not handle lists of tables. This means that a CDF can only have one occurrence of a table type.

---

#### 4.14 Building and Testing

---

Assuming the dui source directory has been checked out, cdfdb source is located under ~user/dui/src/cdfdb. Before anything can be compiled, the Makefile needs to be made. To do so, type in:

```
xmkmf -a
```

After the Makefile has been made, you can make each application individually (i.e. make 843CDFtoDB) or make them all (make all). With a "make all", libcdfdb.a will be installed in ~user/dui/lib and the applications will be in the ~user/dui/src/cdfdb. To remove all the .o files and the executables, type in:

```
make clean
```

To install the CDFtoDB programs in ~gatec2/bin, type in:

```
make install
```

One thing to remember. If you added any .cc file and you need to modify the make in order to build it, make sure you modify the Imakefile. After you have done so, you need to do a xmkmf -a to make the Makefile again.

---

#### 4.15 CDFDB Unit Testing

---

To test CDFtoDB programs, use the -t to view the trace output. For example, if you want to see if an 824CDF will process and commit to the database, type in:

```
cat 824CDF|824CDFtoDB -t
```

For testing 840DBtoCDF(), type in:

```
test -d4 -t
```

For testing 850DBtoCDF(), type in:

```
test -d5 -t
```

---

#### 4.16 System Testing

---

Refer to the GATEC system test procedures.

---

#### 4.17 System Install

---

Refer to the GATEC installation instructions.

---

#### 4.18 Diagnostic Error Messages

---

When a CDFtoDB program fails for one reason or another, mail is sent to a person responsible for correction. The mail message contains a the original X.12 transaction, the CDF created from this transaction, and the error produced by the CDFtoDB program. To change the recipient(s) of these mail messages, the inbound Bourne Shell script needs to be modified. For further details on inbound, refer to the Transport section of this document.



---

## SECTION 5 Transport

---

The transport software is located at \$CVSROOT/transport in the GATEC development environment. Since all modules are Bourne Shell scripts they do not need to be compiled; instead, one may simply install them in a desired directory location. The transport subsystem primarily consists of the scripts inbound(1), outbound(1), and the archive mailbox. This page serves as an outline that will first describe the transport's role and an overview of the general approach used. Next the page will focus on the current architecture of inbound(1) and outbound(1). Finally, the page will briefly look at future enhancements and alternative solutions.

---

### 5.1 Transport Overview

---

The GATEC transport system transmits or receives the Electronic Data Interchange(EDI) data to/from the Electronic Commerce EDI Hub (ECEDI) for the GATEC application. The transport system also translates the EDI data to/from the Common Data Format (CDF) used in the GATEC application using the Government Standard Translator (GST). When the GATEC application releases an EDI transaction, it queues to the outbound queue, by using the lpr(1) command with a CDF file. The lpd(8) invokes the outbound(8) script that routes the CDF file to the correct GST translation, where the GST translator converts the file into an EDI message. Finally outbound(1) script mails the EDI message to the ECEDI hub for delivery. Sendmail(8) places an inbound edi message on the inbound queue via lpr(1). Sendmail accomplishes this by using an alias for the site\_id on the machine i.e.: f33601: archive, "| /home/gatec2/bin/input". The script input(1) finds a queue with space. If space cannot be found, the message returns to the hub and will be tried again later. When lpd(8) starts inbound(1) script, the first the script performs a 997 syntax check on the EDI message using the GST(1) translator, and follows with an 824 semantic check, if the EDI message passes both checks. The script again calls the GST(1) translator to convert the EDI message into a CDF file. Finally the script calls the appropriate CDFtoDB to insert the CDF file into the database. Should the database be down the inbound(1) script will retry inserting the CDF file using the at(1)

command with the cdfretry(1) script at a later time.

Using a standard mailbox named archive accomplishes the task of archiving both inbound and outbound EDI messages for the GATEC system. Simply placing archive on CC: line or including it within an alias places messages into the archive. Anybody can review (but not update) the archive mail box using any standard electronic mail user agent. Each day the archive mailbox gets rolled over to an archive directory called ~archive/archive. Periodically the ~archive directory gets compressed to recover space. Finally the older archives will move over to a permanent storage medium and be removed from the system.

---

## 5.2 Transport Approach

---

The decision to use Bourne shell scripts and existing system utilities like lpd(8) and sendmail(8) in the transport subsystem came as a result of several factors. The biggest factor in using the Bourne shell, was the dynamic nature of the GATEC project itself. As other GATEC components evolved, the transports requirements changed. Frequently the transport subsystem served as the warning system for errors either going into or coming from the GATEC system. Since the transport system didn't manipulate either the EDI or CDF data, but rather functioned as switch or pipeline, made Bourne shell an easy choice. The election of Bourne over kinds shell's was simply makes it the most portable across UNIX platforms. Execution speed of the transport is not a critical factor, since it is not an interactive process with live users awaiting any update.

The approach to use lpr(1)/lpd(8) as the queuing mechanism was three fold, first a unique spooling/queuing system didn't not have to be written. Second managing the lpd(8) spooler should be known to most system administrators and require no special training. In fact, one could say it's easier, since this "printer" never needs paper. In the event of a system V port, the same approach can be used, by only changing "printer" configuration.

The ECEDI approach of enclosing EDI messages within email envelopes made sendmail(8) a natural choice. Since it is the mail transport agent that comes with most UNIX platforms. The GATEC transport subsystem can use the sendmail(8) infrastructure without developing any special code. In the event of moving the GATEC transport system to another platform that has a different mail transport becomes a trivial exercise. The system administrator doesn't require any special training, since managing sendmail is usually a part of the normal duties.

Using a standard mailbox for archiving came as a natural extension of the ECEDI approach to electronic mail enabled EDI. Archiving used the existing electronic mail infrastructure, to fulfill the archiving requirement with little programming effort. The electronic mail header already contains to/from destinations, timestamping, and unique message id, necessary for archiving. Finally using the standard electronic message format allows anyone to use their favorite email user agent(or none) to look at the archive data, without requiring any special training.

---

### 5.3 Addressing

---

The email header information is generated by GATEC at the Site.

To:  
<@ec099.llnl.gov/PN=Joan.Dennis/DD.ID=jomarcomp/O=ATTEDI/@sm2att.llnl.gov>  
Bcc: archive,archive@ec099.llnl.gov  
Subject: 997:F3360193Q1911001

By reading this header, you can tell that this email message header was generated on the Wright-Patterson Site IGP as a X12 Functional Acknowledgment (997) in response to a quote given to Wright-Patterson (F33601) in response to Request for Quote #93Q1911, line item #1 submitted by a vendor (jomarcomp) using the ATT VAN, which is connected to the Livermore Hub using X400 mail.

Following is a detailed decomposition of each field on each line.

Line 1:

In the To: line, the @ec099.llnl.gov field is the Internet mail address of the Livermore Hub. All mail generated by the GATEC system sends outgoing mail to the Livermore Hub via the Internet using SMTP.

The last field in the mail header (@sm2att.llnl.gov) is routing information used after the *SMTP* mail arrives at the Hub. Since we know that ATT uses X400-based mail, we have to send the incoming SMTP mail to the *SMTP* gateway for conversion to X400-based mail. The name sm2att is the identification for the *SMTP*-to-X400 gateway for ATT.

The remaining fields only appear on mail messages intended to be sent to X400-based VANs. They are defined by the X400 mail standards and are further defined in the Retix X400 gateway

software description. The *DD.ID=* field is the X400 *Domain Define.Identifier* field, which is assigned to the name that the vendor electronically registered under. The *PN=* field is the X400 *Personal Name* field and is the name of the contact person at the registered vendor. Therefore, */PN=Joan.Dennis/DD.ID=jomarcomp* indicates that the vendor to whom this mail message is directed is JOMAR Computer Supplies, 2106 Winslow Drive, Orlando, FL. The sales contact at that address is Joan Dennis.

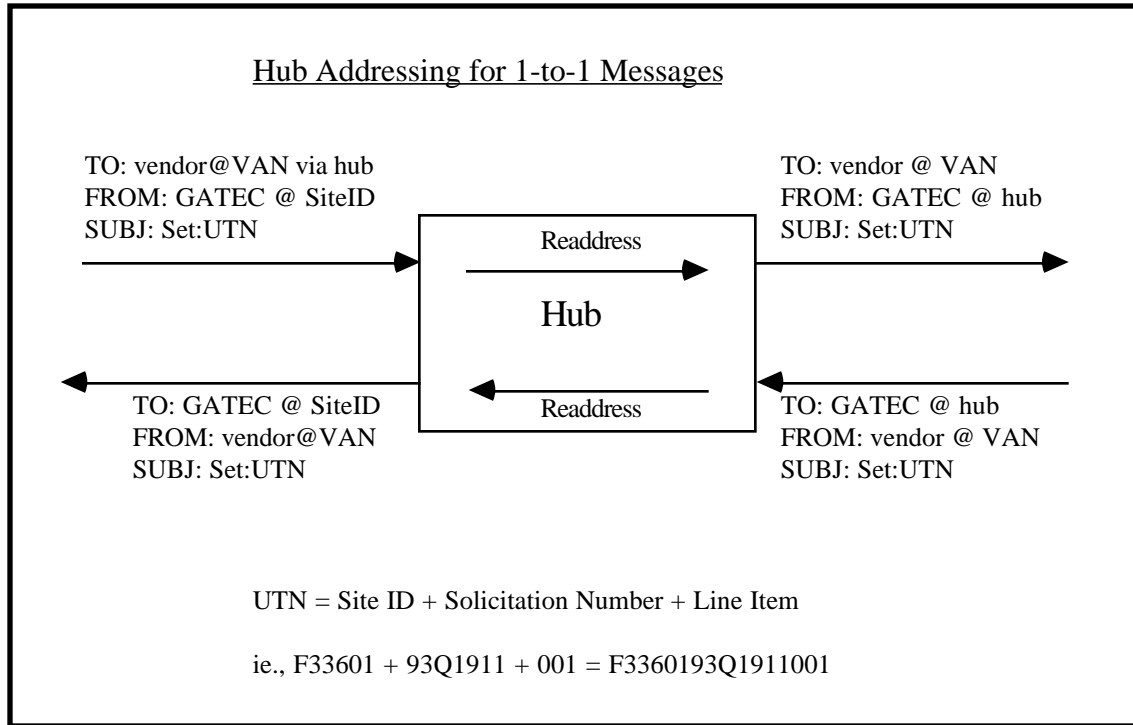
Line 2:

The *Bcc:* line (Blind Carbon Copy) indicates that a complete copy of this message is to be sent to two additional mailboxes; one which is named *archive* at Wright-Patterson and one named *archive* at the Livermore Hub (ec099.llnl.gov). This allows both the Site IGP and the Hub to archive all mail traffic going through the GATEC system as well as the Hub for later audit purposes. Incidentally, it is the GATEC software that creates this line that would change to add DAASC to the mail distribution.

Line 3:

The *Subject:* line is used for email routing and transaction recognition at the Hub. The Hub actually readdresses mail sent from the VANs based on the contents of the Subject line. There are two fields in the Subject line; the X12 transaction set number and the Unique Tracking Number. The 3 digit X12 transaction number (840 for Request for Quote, 843 for Response to Request for Quote, 850 for Purchase Order or Delivery Order, etc) allows the Site IGP to recognize the transaction type before translating it in preparation for loading it into the local database. The Unique Tracking Number is composed of the concatenation of the Buying Site Id (AKA the DoD Activity Address Code - the DODAAC), the Solicitation Number and the Line Item Number. The Site ID (Wright-Patterson ASC/PKW is F33601) is used for routing on mail received from a VAN on its way to the Buying site. This is shown in following figure.





### Hub Addressing for 1-to-1 Messages

## 5.4 Outbound

### NAME

outbound converts CDF files into outbound edi messages.

### SYNOPSIS

outbound

### DESCRIPTION

outbound is invoked by lpd(8) daemon after an outbound CDF file is placed in the queue. outbound first determines the type of CDF file it received and invokes the correct translation that turns the CDF into a x12 message. After the GST translates the CDF file into x12. outbound uses the GST to syntax check the x12 message using the same 997 translation that the inbound(1) uses. If the message fails syntax check, all the data including the 997 output is mailed to gatecmgr. Otherwise outbound shells the output from the GST and checks the return code. If output exits non-zero a

message containing the CDF file and GST output is sent to the gatecmgr mailbox for correction.

#### INTERNAL DESCRIPTION

Determine where to find translation table directories, where to place temp files and build environment.

Extract the type of CDF file from the %purpose field in the CDF file.

Change directory to the correct translation for this CDF file.

Translate CDF into EDI message contained within a mail envelope and the mail transport command.

Change directory to the 997 translation.

Syntax check the EDI portion with the GATEC 997 translation (syntax check) using GST.

Shell the output file from the translator, since it contains the mail transport commands.

Clean up and exit.

#### NOTE:

After each of the above steps, the script verifies exit status for correctness. Anytime the script encounters an error for instance, there is no output from the translator, or a translation table doesn't exist. The CDF file and any available temp files or other information is mailed to the gatecmgr mailbox for manual correction.

#### SEE ALSO

GST(1), BUGS

Shell metacharacters sometimes explode the GST output.

---

## 5.5 Inbound

---

### NAME

inbound converts site received edi email messages into database

records

## SYNOPSIS

inbound

## DESCRIPTION

inbound is typically invoked by lpr(1) whenever an email message is delivered to a gatec site. inbound will use the GST to generate an 997 acknowledgment if the received message requires one. If the incoming message passes the 997 syntax check, next inbound will again call the GST check the semantic content and produce an 824 rejection, should the semantic inspection fail. The last call to the GST translates the inbound edi message into the CDF format used by the gatec project. Finally the inbound invokes the appropriate CDFtoDB database insert program. If the CDFtoDB program exits with database not available error. inbound calls at(1) to invoke cdfretry(1) in one hour to attempt to insert the CDF file into the database. When inbound detects an error anywhere during the process, a message and the incoming edi message gets sent to gatecmgr for human correction.

## INTERNAL DESCRIPTION

Determine where to find translation table directories, where to place temp files and build environment.

Determine EDI message type from EDI and message originator from mail header.

if Message type equals 997 or 838 goto alltocdf translation.

Change directory to the 997 translation (syntax check).

Syntax check the EDI portion with the GATEC 997 translation using GST.

Send 997 acknowledgment back to originator.

If message doesn't pass syntax check exit

Change directory to the 824 translation (semantic check).

Syntax check the EDI portion with the GATEC 824 translation using GST.

If message doesn't pass semantic check exit

Change directory to the alltocdf translation (convert edi to CDF).

Translate EDI message to CDF file.

Select appropriate CDFtoDB insert program based on EDI message type.

Execute CDFtoDB program. If database down invoke cdfretry using at(1) later.

Clean up and exit.

#### NOTE:

After each of the above steps, the script verifies exit status for correctness. Anytime the script encounters an error for instance, there is no output from the translator, or a translation table doesn't exist. The CDF file and any available temp files or other information is mailed to the gatecmgr mailbox for manual correction.

#### SEE ALSO

GST(1), input(1)

#### BUGS

SunOs 4.1.3 lpr(1) has an undocumented limitation of 1000 files that it can spool at one time. input(1) works around this limitation by retrying one of 3 input queues when it encounters an error trying to lpr to an inbound queue. In the event that input(1) fails, the message returns to the hub and is retried later.

---

## 5.6 Transport Support Software

---

The following software supports transport activities at the site  
NAME

---

### 5.6.1 input

---

NAME

input places message on inbound queue

SYNOPSIS

input [ input ]

DESCRIPTION

sendmail(8) invokes input when an incoming edi email message arrives for the GATEC system. input places incoming mail messages on an available lpr(1) queue. This script works around a SUNOS limitation of a 1000 files on the same queue. Input tries an inbound queue, if it gets an error return it will try the next, until it exhausts the list. If it fails to insert the inbound mail message on any queue it exits non zero. When sendmail(8) receives a non zero error back from input it returns the undelivered message back to the ECEDI hub, where the message will be retried at a later time. Presumably after the inbound queues have drained some.

SEE ALSO

inbound(1), lpr(1), lpd(8), sendmail(8)

BUGS

Currently configured for 3 inbound printers.

---

## 5.6.2 newsyslog

---

### NAME

newsyslog - syslog and gatec archive mover

### SYNOPSIS

newsyslog [ newsyslog ]

### DESCRIPTION

cron(8) executes newsyslog everyday around midnight to move the sendmail(8) log file and the archive mailbox to the archive directories under ~archive. After the log files and archive has moved the files are erased and sendmail(8) is restarted. As a secondary function of newsyslog, the script goes to ~gatec2/tmp and removes old temp files that are currently being kept around for a period of time for debugging purposes. This function will go away in some future release.

### SEE ALSO

cron(8), sendmail(8), transport(1)

---

### 5.6.3 cdfretry

---

#### NAME

`cdfretry` retry to insert cdf file into the database

#### SYNOPSIS

`cdfretry [ cdfretry CDF_FILE attempts ]`

#### DESCRIPTION

The `at(1)` batch processing daemon invokes `cdfretry` at the scheduled time to again attempt to insert an already translated x12 transaction in CDF format into the database. Presumably because the database was down in the previous attempt(s), `cdfretry` expects the name of the CDF insertion program for the type of CDF file contained in `CDF_FILE`. If `cdfretry` succeeds it removes `CDF_FILE` and quietly goes away. Otherwise `cdfretry` bumps up the attempts and re-queues itself with `at(1)` to try again later. After 10 tries, `cdfretry` will send the `CDF_FILE` to `gatecmgr` for manual processing.

#### OPTIONS

name of the CDFtoDB executable  
`CDF_FILE` temp file containing a translated x12 message into common data format.  
attempt number of tries so far to insert this record into the database.

#### SEE ALSO

`at(1)`, `inbound(1)`

---

## 5.7 Future Enhancements

---

The transport scripts need some cleanup, the first thing to do would be to compile the GST(1) translations and load them with the translator engine module to produce single translation binaries for each translation. Once the translation is an executable, the transport scripts can get rid of the overhead of having to change directory for each translation. Next the scripts need to be better modularized and use some shell procedures to reduce some of the redundant code. Finally, some changes to the code to make it a little more

readable and perhaps a bit more understandable. Another nice feature would be a single utility to properly setup the GATEC environment with all the necessary things.

In the area of consistency and accuracy, the transport needs a better coupling with both the database and the archive to insure that all messages are correctly sent or received and errors accounted for. The current architecture lends itself to a fire and forget solution. After the GATEC application releases a CDF file for conversion and transmission, the transport doesn't update the database of success or failure. Also the transport system doesn't verify archive records with the database on incoming transactions. Perhaps an future redesign of the GATEC system could combine the CDFtoDB and transport functionality into single inbound and outbound daemons that could have a tighter coupling with the database and eliminate the need for the CDF<->EDI translations and move to DB<->EDI translations.

---

## 5.8 Configuration Dependencies

---

Although the GATEC transport subsystem lacks a central configuration file to pick up it's environment. Both inbound(1) and outbound(1) initialize the same way. The top of each script holds all the path dependent parameters, in which the scripts sets into temporary variables. The scripts get the gatec2 home directory from the passwd(5) file. The scripts create their temp files under the ~gatec2/tmp directory. The translations used by the GST(1) are found under the ~gatec2/lib/gst directory. When the script encounters an error condition it sends as much data as it knows about to the gatecmgr id for manual correction. In order to configure a new address, change the OOPS\_ID variable to the new value.

For installation instructions for the GATEC transport subsystem refer to the *GATEC Operations Manual*.



---

## SECTION 6 GATEC 2 Test Matrix

---

The GATEC system is tested with the matrix shown below. If all testing criteria are met, the GATEC software is said to be operating nominally. Observe that this implicitly verifies that the software modules described in sections 1-5 are executing correctly.

---

### 6.1 The Matrix

---

<b>GATEC TEST PROCEDURE (version 1.4)</b>	<b>OK</b>
<b>NOTE:</b> It is suggested the following tests be performed on a recently exported copy of the current WP database	
Issue 5 RFQ's	
Fill in manufacturer field (make sure it sticks)	
Fill in part number field (make sure it sticks)	
Make sure RFQ date is current date	
Make sure priority class matches day on street time PRI 1-3 response date 4 days 4-9 response date 5 days rest response date 5 days	
Make sure delivery date is appropriate to the priority PRI 1-3 7 days delivery 4-9 delivery in 30 days rest delivery in 30 days	
Make sure delivery date/response date does not fall on weekend or holiday	
Make sure item description has no missing text	
Make sure RFQ number exists and correct	
Make sure line item number exists and is correct	
Make sure requisition number exists	

Make sure priority exists	
Make sure stock number exists	
Make sure FSC number exists	
Make sure suffix exists	
Make sure estimated price exists	
Make sure quantity exists	
Make sure unit of issue exists	
Make sure extended price exists and is calculated correctly (quan*price)	
Make sure shiptozip exists	
Make sure Addresses is set to public	
Make sure FSC is editable and changes stick	
Make sure quantity is editable and changes stick	
Make sure response date is editable and changes stick	
Make sure delivery date is editable and changes stick	
Make sure additional clauses is editable and sticks	
Toggle item description upload box to make sure when enabled CDF containing item description is placed on bcasitemupload queue	
Test auto disable of item description upload by making sure auto upload is disabled when item description is read in which had already been uploaded by GATEC.	
Review item description CDF to insure it is correct	
On 840 make sure BQT segment indicates proper response date	
On 840 make sure BQT segment indicates 00 (for new RFQ)	
On 840 make sure DTM segment indicates correct delivery date	
On 840 make sure P01 segment has correct quantity and unit of issue	
On 840 make sure PID segments have correct item description	
On 840 make sure P01 segment has correct manufacturer	
On 840 make sure P01 segment has correct part number	
On 840 make sure P01 segment has correct FSC, and stock number	
On 840 make sure REF*65 segment has correct UTN	
On 840 make sure GS segment has current date	
On 840 make sure segment count in SE segment is correct	
Make sure 840 is received by test vendors on our VANS and can be read using their software	
Issue several 840s that are specifically directed. Try combinations of issuing to PUBLIC as well as to other directed cage codes. Make above checks on those 840's and insure they are correctly delivered.	
Try separating cage codes for directed RFQs with white space, comma, or carriage return. All should be recognized	
Try specifying bogus cage codes for directed RFQs. Make sure system recognizes the bad cage codes	
Make sure no fields can be edited while RFQ is under open	
Using send864, send in at least three messages (>30 lines) per RFQ	
Make sure U appears by RFQ's messages where sent to on workload screen	
Make sure all messages can be viewed, and get placed in read category	

Make sure each message can be responded to, using respond	
Make sure each of the sent messages can be reviewed under sent	
Make sure each received message can be placed in needs action	
When a message has been placed in needs action, make sure N appears by RFQ number on workload screen	
Attach 5 notes, using compose, to each RFQ. Each note should be at least 30 lines long	
Make sure each of the notes can be reviewed under the notes category	
Make sure 864 responses get to LLNL test vendor	
On 864 make sure DTM segment has today's date	
On 864 make sure REF*65 and REF*DX has UTN number	
On 864 make sure REF*IX segment has correct line item number	
On 864 make sure MSG segments have the correct message text	
On 864 make sure the SE segment has the correct segment count	
Choose an RFQ to amend (under open), change some fields, and send the RFQ out again (via confirm amend)	
On 840 make sure BQT segment indicates amend (01)	
Make all other standard 840 checks mentioned above, insuring the amended information is on the 840	
Using sendThemAll make standard bids, bids with terms, bids with GSA contract numbers, alternate bides, bids indicating can't quote, and bids with nte text for the 5 RFQ's that were issued	
Use the close utility on delphi to move RFQ's from open to closed	
For each closed RFQ make sure there has been no change in message status when the RFQ was moved from OPEN to CLOSED	
For each RFQ go to the REVIEW QUOTES screen	
Make sure RFQ number is correct	
Make sure line item number is correct	
Make sure requisition number is correct	
Make sure FSC code is correct	
Make sure suffix is correct	
Make sure priority is correct	
Make sure stock number is correct	
Make sure estimated price is correct	
Make sure quantity is correct	
Make sure unit of issue is correct	
Make sure extended proce is correct	
Make sure item description is correct	
Make sure all fields are read only	
Make sure all quotes sent in appear	
Make sure a columns align	
Make sure payment column is populated for bids with terms	
Make sure proper boxes indicating flag types are checked (e.g. G for GSA contract)	
Make sure proper flags are used (e.g. G for GSA contract, etc) under flag column	

For each quote, go to the REVIEW QUOTE screen	
Make sure RFQ number is correct	
Make sure line item number is correct	
Make sure stock number is correct	
Make sure estimated price is correct	
Make sure FSC is correct	
Make sure SIC (if any) is correct	
Make sure Item description is correct	
Make sure quoting vendor cage code is correct	
Make sure quote effective date is correct	
Make sure quote expires date (if any) is correct	
Make sure vendor name is correct	
Make sure unit price is correct	
Make sure quantity is correct	
Make sure unit of issue is correct	
Make sure extended price is correct	
Make sure delivery date is correct	
Make sure payment percent, days, and net are correct if vendor specified terms	
Make sure variation (if any) correct	
Make sure FOB (if any) correct	
Make sure flags specified are correct	
Make sure quote description matches text on input 843	
Make sure FSS contract number and expiration date are correct if GSA contract has been specified	
Amend RFQ and send in more bids. Make sure existing bids (received before amendment) are marked with the flag indicating received before amendment	
Make sure nte text shows up correctly (and is scrollable) on quotes that have attached nte text	
Make sure alternate bid flag shows up when alternate bids have been received	
Make sure bids which come in indicating unable to quote are so indicated on the REVIEW QUOTES screen	
Make sure bids coming from cage codes who have sent in unread 864 messages are so marked with the M flag in the REVIEW QUOTES screen. Also make sure M goes away when messages are read	
Make sure checking for govt. password and cage work (i.e. send in bids with bad cage code and or bad govt. password	
Insure 824 which is generated is correct	
Select a non gsa bid with no terms and go to MAKE AWARD	
Make sure correct automated PIIN number comes up	
Make sure RFQ number is correct	
Make sure line item correct	
Make sure contract field blank (non gsa)	

Make sure date field has current date	
Make sure order statements read EX IN SI GU (non gsa)	
Make sure quantity is correct	
Make sure unit of issue is correct	
Make sure unit price is correct	
Make sure transaction totals are correct (unit price * quantity)	
Make sure delivery date is correct	
Make sure awardee's name is correct	
Make sure first 5 characters of bcas vendor code is correct	
Make sure FOB point reads D (non gsa)	
Make sure no variation	
Make sure no payment percent, days, or net if no terms specified	
Make sure DO rating is c9b	
Make sure negotiation authority is 0301	
Make sure competition code is y	
Make sure confirmation field is blank	
Make sure BSP field has correct buyer	
Press MAKE AWARD; if successful next closed RFQ will display its bids	
Next make an award to a bidder who has specified bids	
All checks should be the same as mentioned above, except for the following:	
Make sure payment percent, days, and net fields have the correct data specified	
After doing all checks, press MAKE AWARD again.	
Next, make an award to a bidder who has specified a GSA contract number	
All checks should be the same as mentioned above except for the following:	
Make sure correct GSA contract number displayed on award screen	
Make sure PIIN number used is of GSA type	
Make sure order statements read IN SI	
Make sure negotiation authority reads INTG	
After doing all checks press MAKE award again	
Next using downLoadPiins "unuse" a prior used non GSA piin. Make another award and make sure that unused piin is used	
Do the same thing for a GSA piin.	
Make awards for RFQs who have had their quantity changed before issue, after issue (amendment), and during award. Make sure the P/Q flag is correct in the BCAS cdf for quantity increase (P) and quantity decrease (Q).	
Try awarding to bidder who has submitted bids from 2 different VANs. Make sure 850 goes to the right VAN	
Next using downLoadPiins, mark all non GSA piins as used. Make another award and make sure the award piin box is blank and user is allowed to input piin. Make sure award can commit	
Do the same thing for a GSA piin	

Try redirecting a closed RFQ and insure that the RFQ "goes away" from the closed workload screen	
Try HOLDING from OPEN/UNISSUED/CLOSED	
Try RE-DIRECTING from OPEN/UNISSUED/CLOSED	
In the case of re-direct make sure re-direction can occur for each of the twelve specified reasons. Make a similar check for HOLD (but hold only has 4 possible reasons)	
On 850 make sure GS segment has current date	
On 850 make sure REF*65 has correct UTN number	
On 850 make sure REF*DX has correct UTN number	
On 850 make sure DTM segment has correct delivery date	
On 850 make sure N1 loops have correct WP contracting, accounting, and delivery addresses	
On 850 make sure P01 segment has correct price, unit of issue, stock number and quantity	
On 850 make sure PID segment has correct item description	
On 850 make sure AMT segment has correct total (unit price*quantity)	
On 850 make sure SE segment has correct segment count	
On 836 make sure P01 has correct price, unit of issue, quantity, stock number and federal supply class	
On 836 make sure BCO segment has correct RFQ issue date and correct RFQ award date	
On 836 make sure REF*65 statement has correct UTN number	
On 836 make sure SE segment has correct segment count	
On 836 make sure GS segment has current date	
Inspect all BCAS CDF's to make sure they will be uploadable	
Verify 997 generated upon receipt of 864 and or 843	
Initiate at least 4 tty clients and have each of them initiating REVIEW RFQ while a PC client starts up, switches category from UNISSUED to CLOSED, REVIEW QUOTES, switches category from CLOSED to OPEN, then REVIEW RFQ. Insure delay time between any one action does not exceed 3 to 4 minutes.	
Verify Acknowledgment checking is working for all 840's,850's, 864's generating during testing	
Attempt to cancel awards. Verify CDF generated is uploadable (and properly queued) and will result in cancel. Also insure 850 CDF will produce proper 850 cancel X12 (code 01 used in BEG segment)	

## Revision History

- 1.1            6/29/93            Added check on response date/delivery date times based on priority level(s)
- Added check to make sure response date does not fall on holiday/weekend.
- Added check for sending in alternate bid, non quoting, and bids with nte segment text
- Added check to make sure received prior to amendment flags working on received 843's after amendment
- Added check on enable/disable item description upload
- Added review item description upload check
- Added test of auto disable for item description upload
- Added test of re-direction for all 12 categories
- Added check of cage code/govt. password
- Added check of 824 generated under cage code/govt. password errors
- Added 997 generator check for 843, 864
- 1.2            8/3/93            Added specific HOLD check. Added checks for both HOLD and REDIRECT from UNISSUED and CLOSED
- Added speed testing using 5 tty clients
- Added constraint to export a copy of WP database before testing
- 1.3            10/4/93            Added check of acknowledgment system,
- Added check for cancel award
- Added check for directed RFQs
- Added check for search on piin/RFQ number
- Added check with test vendors on VANs
- 1.4            11/16/93            Added check to make sure bid with unread 864 marked with M flag on REVIEW QUOTES screen
- Added checks for correct P and Q calculation in BCAS CDF

Added check for submission of quotes and award to vendor who is using 2 VANs





Technical Information Department • Lawrence Livermore National Laboratory  
University of California • Livermore, California 94551